VISILAB
SIGNAL TECHNOLOGIES

Designer's Manual for

Packet Protocol

Used in all AK30/40 and AK50 Moisture Meters and Other Products by Visilab Signal Technologies

PART #700219

Made in Finland
Manual printed in
Pukkila, Finland

2015-04

Copyright (c) 2015 Visilab
Signal Technologies Oy

Visilab Signal Technologies Oy
Address:
Keskustie 15
FI-07560 Pukkila
FINLAND

Tel.:
+358-45-635 4885
Y 0631208-0 ALV Rek.
VAT FI06312080

e-mail:
info@visilab.fi

Mäntsälä Reg. 365.258

www.visilab.fi

CONDITIONS OF GUARANTEE, COPYRIGHT NOTICE AND LIABILITIES OF THE MANUFACTURER

The manufacturer (Visilab Oy) grants a guarantee of two years for the buyer of AK30/40/50 moisture meter from the date of purchase. The guarantee covers all faults and misalignment which are in the equipment at the moment of purchase including those which appear during the guarantee period. The manufacturer is liable of repairing the instrument without cost to the buyer. The manufacturer can ship a new instrument of equivalent value and status if considered as a better solution than repairing. The buyer is liable of paying the freight costs to the factory of the faulty unit. The unit must not be sent to the manufacturer without a permission from the manufacturer. Units sent without a permission will be repaired at the cost of the buyer.

The guarantee does not cover wearing parts, like batteries, lamps or motors. The guarantee does not cover faults caused by errors or neglects of the user nor those faults which are caused by deliberate breaking. The guarantee does not cover faults caused by incorrectly installed cables or conductors. The guarantee does not cover any damages to the user or to any third party independently of the way how the instrument has been used. The guarantee does not cover faults caused by natural phenomena like lightnings or floods, nor user errors like dropping the unit. The guarantee is void if the unit is sold to any third party. All faults which are not covered will be repaired at the cost of the buyer.

If opening of the instrument has been attempted at those parts which are not intended for the user, the manufacturer can refuse to repair or service the instrument. Then the instrument will be shipped back to the buyer at the cost of the buyer. Such parts are the light source, the optical head and parts on the electronics board. The instrument can be opened only strictly according to the instructions in this manual and should not be disassembled unnecessarily.

Copyright (c) 1994 - 2015 Visilab Signal Technologies Oy, All Rights Reserved

Visilab Oy reserves all rights to changes and modifications in the looks, specifications, optical and electronic design, electronic and software interfaces and computer programs, and also the right to change the retail prices of the instrument or its parts without any notice to present or potential customers. All copyrights and design rights belong to Visilab Oy. The PC programs, which have been sold to the buyer, can be used and copied freely for his own use but can not be sold to any third party.

The manufacturer is not responsible for any casualties, damages or accidents which the user has caused directly or indirectly with this instrument, either to himself or to any third party.

Contents

Table of Commands Available for the Traditional IRMA-7 Protocol Used in Many of Visilab's Products....4
1. Introduction....7
Configuring....7
Troubleshooting Hint....7
2. Operating the Slave via Packet Protocol....8
General....8
Passing Commands....8
Frame Data....8
General Commands....11
Calibration and Standardization Commands....40
Data Acquisition Commands....57
Memory Bank Commands....62
Get the Current Burst Size:....80
Text String Commands....86
Appendix 2. Technical Specifications of the Protocol....110
Basic Rules for the Slave....110
Basic Rules for the Master....111
Appendix 3. Examples of the CRC calculation 114
Appendix 4. Delay Analysis for the Packet Protocol....117
Appendix 5. Numerical Values of Commands and Constants Used in the Packet Protocol....119
Index....123

Table of Commands Available for the Traditional IRMA-7 Protocol Used in Many of Visilab's Products

General Commands (decimal and hexadecimal values)

	decimal	*hex*
Get the General System Status I7GSTATUS	76	0x4C
Get the Second System Status I7G2STATUS	86	0x56
Get the Third System Status I7G3STATUS	89	0x59
Read the Filter Characteristics I7GFILTER	50	0x32
Change the Filter Characteristics I7SFILTER	49	0x31
Get the Locking Status I7GETLOCK	53	0x35
Set the Locking I7GAINLOCK	51	0x33
Set the Autoranging I7GAINOPEN	52	0x34
Get the Lamp Status I7GLAMP	74	0x4A
Get the Chopper Speed I7GFREQ	60	0x3C
Get the Usage Counter Hours I7GETUSG	28	0x1C
Set the Terminal Mode ON (Keyboard Mode) I7STERM	47	0x2F
Get the Low Power Mode Status I7GETLPM	37	0x25
Set the Low Power mode ON/OFF, (ON = Low Power Mode, OFF = Normal Mode) I7SETLPM	38	0x26
Get the Voltage Output Source I7GVOUT	88	0x58
Set the Voltage Output Source I7SVOUT	87	0x57
Get the Cooler Enable Status I7GCOOLING	90	0x5A
Get the Cooler Temperature I7GCOOLTMP	93	0x5D
Get the Cooler On/off Status I7GCOOLON	94	0x5E
Get the Cooler Linking Status I7GCOOLINK	95	0x5F
Get the Cooler Status I7GCOOLSTA	97	0x61
Set the Cooling Enable I7SCOOLING	92	0x5C
Set the Cooling Linking I7SCOOLINK	96	0x60
Set the web temperature filter setting I7STLPF	99	0x63
Get the web temperature filter setting I7GTLPF	101	0x65
Set the web temperature offset I7SWEBB	102	0x66
Get the web temperature offset I7GWEBB	103	0x67
Get the head temperature alarm status I7GALM	104	0x68
Clear the head temperature alarm I7CALM	105	0x69

Calibration and Standardization Commands

	decimal	*hex*
Get the Current Material Entry I7GETMAT	14	0x0E
Switch to another Calibration Table in the Library I7SETMAT	15	0x0F
Get the Calibration Mode of the Current Material Entry I7GMODE	16	0x10
Set the Calibration Mode (MULTI/QUICK) I7SMODE	17	0x11
Read the Calibration Table Entry I7RXMAT	27	0x1B
Set the Calibration Table Entry I7TXMAT	26	0x1A
Set the Standard Moisture Value for Standardization I7SSTD	72	0x48
Get the Material Entry Number Used in Standardization I7GSTDM	71	0x47
Set the Offset Value for Standardization I7SSHIFT	67	0x43
Get the Offset Value Resulting from Standardization I7GSHIFT	68	0x44
	decimal	*hex*

Get the Standard Value Set for Standardization I7GSTD	73	0x49
Standardize I7STDZE	69	0x45
Set the Standard Material Entry Number I7SSTDM	70	0x46

Data Acquisition Commands

	decimal	*hex*
Get the Moisture or Other Primary Signal I7MOIST	11	0x0B
Get the Optional Extra Web Temperature I7GWEB2	100	0x64
Start Sending the Head Temperature instead of the Web Temperature I7GETTMP	46	0x2E
Start Sending the Web Temperature Instead of the Head Temperature I7GWEB	48	0x30
Get expansion module signal I7GXMOD	108	0x6C

Memory Bank Commands

	decimal	*hex*
Read the Number of Samples in the Current Bank I7GETDM	35	0x23
Read the Bank Number I7GBANK	55	0x37
Get the Autotimer Mode I7GAMODE	59	0x3B
Get the Autotimer Status I7GETAUTO	43	0x2B
Clear the Current Data Series (or Bank) I7CLRSER	21	0x15
Take a Sample into the Current Data Series (or Bank) I7SAMPLE	36	0x24
Set the Autotimer ON I7AUTOON	41	0x29
Set the Autotimer OFF I7AUTOOFF	42	0x2A
Select the Bank I7SBANK	54	0x36
Set the Autotimer Mode I7SAMODE	58	0x3A
Get the Autotimer Interval in 0.1ms Units I7GETTIM	40	0x28
Get the Current Batch Size I7GBATCH	57	0x39
Set the Current Batch Size I7SBATCH	56	0x38
Set the Autotimer Interval in Seconds I7SETTIM	39	0x27
Get Samples from the Current Memory Bank I7TXSER	20	0x14
Copy the Temperature Series to Bank4 I7COPYT	98	0x62
Set the Current Burst Size I7SBURST	112	0x70
Get the Current Burst Size I7GBURST	113	0x71
Set the Burst Mode I7SBUM	114	0x72
Get the Burst Mode I7GBUM	115	0x73
Get the Burst Mode Item Count I7GBUC	116	0x74
Clear the Burst Mode Item Count I7CBUC	117	0x75

Text String Commands

	decimal	*hex*
Get the Unit for Moisture I7GUNIT	13	0x0D
Get the Current Material Entry Name I7GMATNM	31	0x1F
Get the Current Library Name I7GLIBNM	29	0x1D
Get the Meter's Identifier String 1 I7TEST	10	0x0A
Set the Current Library Name I7SLIBNM	30	0x1E
Set the Unit for Moisture I7SUNIT	12	0x0C
Get expansion module name I7GXNAME	110	0x6E

Special Commands

	decimal	*hex*
Initialize the Profibus DP slave I7DPINIT	65	0x41
Send a Short Pulse to the LED Indicator (if available on the connector panel) I7BEEP	34	0x22
Get expansion module number I7GNXMOD	109	0x6D
Send a command to the expansion module I7SXCOM	111	0x6F
Start Fast Fourier Transform in the Meter I7FFT	83	0x53
Get the Profibus DP Address I7GDPADR	61	0x3D
Set the Profibus DP Address I7SDPADR	62	0x3E
Get the Profibus DP Active Status I7GDPACT	66	0x42
Set the Profibus DP Active I7DPACT	63	0x3F
Set the Profibus DP Inactive I7DPDEACT	64	0x40
No Operation I7NOP	91	0x5B
Get the Switch High Level Setting I7GETHI	32	0x20
Set the Switch High Level Setting I7SETHI	18	0x12
Get the Switch Low Level Setting I7GETLO	33	0x21
Set the Switch LowLevel Setting I7SETLO	19	0x13
Prepare Instrument for Parameter Download I7CDPREP	106	0x6A

1. Introduction

This document instructs you on the details of a packet protocol used by Visilab's instruments in RS232 and RS485 type communications in local and long distance operation in industrial environments. You can create a new kind of data acquisition system using the commands available in your instrument. Since you already have the slave part ready and functional, testing and developing the master protocol to your industrial PC is half done. The rest of the job is covered by this manual to complete the protocol in all respects. This document covers all necessary aspects of the protocol for full implementation. By using this as a backbone, you can design your own error-tolerant, fast and economical protocol. This has been used in very small embedded systems having a weak CPU and less than 32 kb total code space for data acquisition, mathematics, user interfaces and control plus communications. Yet the result has been validated to work according to the protocol's requirements.

Configuring

To configure your packet protocol slave you need to set the slave address and know which model it is. This defines what kind of input data can be retrieved from the slave and what kind output data can be sent to it. The most important specifications of **meter** as a slave are the following:

o 127 bytes maximum packet size, variable size packets sent by master, variable size in packets sent by slave. With some commands, an expected size of a reply packet
o master address always 0
o CRC checking of packet integrity
o time-out period for slave: 50 ms
o time-out period for master: 500 ms
o slave address: 1 (initially, can be changed with the aid of PC interface to any value between 1 and 255)
o transmission rate: 9600 baud or HIGH-SPEED 38400 baud, TOP SPEED 115200 baud RS232
o global commands: a single true global command is the multi-ESC + "x" + "1" command which moves all slaves to packet protocol without exception. The number of ESC's must be at least 8

Troubleshooting Hint

The **RS485** cable has two wires only plus a protecting shield in the minimum configuration. If the slave refuses to operate at all, swap the two pins in the cable connector. That should do it. Refer to Appendix 1. for schematics of the distribution box.

The meter has a DATAEX indicator lamp on the middle CPU board which is visible when the unit access cover is opened. When the lamp is off, the **Profibus DP** slave has been accepted as a correctly configured slave and data exchange is possible. If the lamp is ON, the slave is off-line and does not belong to the DP network. The master will always indicate successful slave parametrization and configuration. Refer to the DP master's manuals for operating the master. Refer to the **AK50** User's Manual for locating the indicator lamp.

2. Operating the Slave via Packet Protocol

General

The **Packet Protocol** is based on slave polling and passive replies. Slaves never send anything by themselves. This protocol is highly efficient specifically due to its error tolerance and dense data packets and avoiding useless packet sending. It is also sensitive in detecting potential data transmission errors and practically always detects incorrect frames by the CRC check and other inherent frame checks. The CRC packed by the sender is checked against the CRC of the receiver for the same packet to make sure no bits are reversed. Also the packet length and recipient address are checked by the slave. This protocol allows simultaneous listening of all slaves which ignore the rest of the packet when they detect that it is not for them.

Each slave to be connected to the RS232/RS485 bus has a just a few special features which makes it different from other slaves. These features are the address and the model. The master does not recognize the slave model but it should be inquired and the user should determine if it is acceptable. The adoption of incorrect models may cause unexpected delays to communication since not all models accept all commands. However, all most important commands are common to all.

Passing Commands

The procedure for sending any of these commands is the following. Note that it is required to send the command **only once** to a slave. Sending a command repeatedly will redo the same thing and possibly overload the slave. Do not send a command without data if the command requires it. Refer to Appendices for details. Incomplete commands will be interpreted as is as long as they are according to the packet frame.

A calibration table change is done immediately and the next moisture values will be linearized according to it. There is a short calculation period of less than 1 ms associated with the table change, during which time the acquired moisture value may be incorrect. That is however, usually not important due to the nature of the situation where this command is used.

Frame Data

The data in the frame should be handled in the following way when commands are sent to the slave. The frame minimum length is five bytes if no data part is carried (**len** = 0).

o max 127 bytes frame (bo1...bo127):

- o 1. byte bo1 : address **adr** of target slave
- o 2. byte bo2 : command **com**
- o 3. byte bo3 : length of data part **len** in bytes
- o 4. byte bo3 : data
- ...
- o nth byte bon: data
- o len + 4 byte : CRC high byte
- o len + 5 byte : CRC low byte

and the reply packet frame is similar:

o max 127 bytes frame (bo1...bo127):
- o 1. byte bo1 : address **adr** of target slave
- o 2. byte bo2 : return status of slave **sta**
- o 3. byte bo3 : length of data part **len** in bytes
- o 4. byte bo3 : data
- ...
- o nth byte bon: data
- o len + 4 byte : CRC high byte
- o len + 5 byte : CRC low byte

The data bytes are sent without any extra delays to complete the packet. If delays are detected at either end, that will cause error detection and possibly resending of command. .

The general structure of the frame is the following:

Example Command Explanation Header:
I7COMMAND DD 0xHH

```
command sent by master:
-----------------------------//-------------
|adr|len|com|d00|d01|d02|d03|// dnn|crh|crl|    frame type
-----------------------------//-------------
  1   2   3   4   5   6   7              byte

reply sent by slave:
-----------------------------//-------------
|adr|len|sta|d00|d01|d02|d03|// dnn|crh|crl|
-----------------------------//-------------
  1   2   3   4   5   6   7              byte
```

The fields filled with some bold identifiers mean that they are already fixed in their use. The first three bytes are fixed in position relative to the packet and the CRC count bytes are always as last bytes in the frame. Deviation from this arrangement causes a nonfunctional protocol.

The short names of various bytes above and later on, to describe the bytes are below.

adr	target address of slave (1 to 255, zero is not allowed when master is sending, zero must be used when slave is sending)
com	command - user defined, e.g. I7GSTATUS as a byte
sta	status byte, unsigned char
len	length of data part in bytes, max 122
data	general data, not defined more precisely at this stage
s	string data, one character of it, may be an end marker also (zero)
c	a single character data, unsigned char
i	integer data, lower or upper byte
l	long integer data, any of the four bytes
f	floating point data formatted as two integers, the whole part and the fractional part. Typically the data is put into four or two bytes
0	zero, set in command, returned in set data

1	one, set in command, returned in reply
(empty)	field not used, ignore any return values
mw	moisture whole
mf	moisture fraction
tw	temperature whole
tf	temperature fraction
xw	expansion module signal whole
xf	expansion module signal fraction

In addition, each command in the next paragraphs has been classified as to its **frame type.** The possible types are the following. There are some special commands which do not fall into these categories due their more complicated data structures in the frames.

setchar	one byte data part, setting some data item or option
getchar	one byte data part, getting some data item or option
setcom	zero byte data part, a simple direct command only
setfloat	four byte data part, setting some data item
getfloat	four byte data part, getting some data item
setstr	N byte data part, setting some text string item of predefined length
getstr	N byte data part, getting some text string item of predefined length
special	special type for complicated data transfer

General Commands

Get the General System Status:
I7GSTATUS 76 0x4C

```
command sent by master:
---------------------
|adr|len|com|crh|crl|   getchar
---------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
------------------------
|adr|len|sta|d00|crh|crl|
------------------------
  1   2   3   4   5   6   byte

len = 1
```

The data byte is bit-controlled and can have the following values.

bit	name	value 0	value 1
0:	Low power mode	**Normal oper.**	Low power mode
1:	Keyboard mode	**Packet mode**	Keyboard mode
2:	Calibration mode	QUICK	**MULTI**
3:	Autotimer mode	**Batch**	Auto (continuous)
4:	Autotimer:	**OFF**	ON
5:	T-autotimer:	**OFF**	ON
6:	Gain locking:	**OFF**	ON
7:	Lamp OK:	lamp fault	**lamp OK** (note 1.)

Flag settings marked with **bold** are default values for general use.

Note 1. If your moisture meter supports by hardware this feature, the flag is valid, else it will always be marked as OK, even in case of lamp failure. However, it is possible to identify from an operating meter's moisture signal that its light source is not working as the signal changes with large amplitudes (like between +/-100% moisture) without any correlation of the target. Refer also to the next command.

Get the Second System Status:
I7G2STATUS 86 0x56

```
command sent by master:
----------------------
|adr|len|com|crh|crl|   getchar
----------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
------------------------
|adr|len|sta|d00|crh|crl|
------------------------
  1   2   3   4   5   6   byte

len = 1
```

The data byte is bit-controlled and can have the following values.

bit	name	value 0	value 1
0:	burst mode	off, normal oper.	**on, BURST mode**
1:	analog output	**moisture**	web temperature
2:	quiet booting	**no**	yes
3:	linked autotimers	**no**	yes
4:	web OK, no break	FALSE	**TRUE (depends on web)**
5:	session start phase	**OFF**	ON
6:	reflective surface	**false**	true
7:	dark surface	**false**	true

Flag settings marked with **bold** are default values for general use. The burst mode is a special operating mode available for piecewise web measurements. The analog output can be configured in the menu system (Vscales). Quiet booting is required when using a LAN and RS485. The setting is done in menu 5 Services. The autotimers for moisture and web temperature can be linked with the temp as a slave in Unit menu - temp series. The web OK bit is an attempt to indicate web breaks (active low). They are interpreted as ultimate limits of darkness or reflection for at least three seconds duration. The bit will be automatically updated. The session start bit is set if the meter has just booted and this will take some 30 seconds. Reflective and dark surface indicators will be used momentarily in normal operation if conditions change. If the bit is set for a prolonged period, something may be wrong. Refer also to the command before this one.

Get the Third System Status:
I7G3STATUS 89 0x59

```
command sent by master:
---------------------
|adr|len|com|crh|crl|   getchar
---------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
------------------------
|adr|len|sta|d00|crh|crl|
------------------------
  1   2   3   4   5   6   byte

len = 1
```

The data byte is bit-controlled and can have the following values.

bit	name	value = 0	value = 1
0:	cooling enable	off	**on, cooler enabled**
1:	**cooling status**	**FAILURE, more air is needed**	**OK**
2:	cooler linking	**no**	yes
3:	web break suspicion	**no break**	yes, a break is suspected
4:	web temperature filter	OFF	**ON**
5:	**overtemperature alarm**	**OFF, OK**	**ON, overheating of head**
6:	COMPOSER	inactive	active
7:	Expansion module	none	installed

Flag settings marked with **bold** are default values for general use. **The cooler settings are related to the meter hardware and apply only if a cooler exists.** The web break suspect bit is an attempt to indicate suspected web breaks when the gain locking is used. Refer also to the two commands before this one. The COMPOSER bit tells if the calibration expert system is installed into your moisture meter. If it is not supported, this bit reads zero.

Read the Filter Characteristics:
I7GFILTER 50 0x32

```
command sent by master:
----------------------
|adr|len|com|crh|crl|   getchar
----------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
------------------------
|adr|len|sta|d00|crh|crl|
------------------------
  1   2   3   4   5   6   byte

len = 1
```

The data byte indicates the inquired setting and may have any of the following values:

OFF	120	0x78
FAST	121	0x79
MEDIUM	122	0x7A
SLOW	123	0x7B
SPECIAL	124	0x7C
BOX	125	0x7D

Refer to User's Manual for more details and effects of the filter setting. This command can not be used with IRMA-7 model A.

Change the Filter Characteristics:
I7SFILTER 49 0x31

```
command sent by master:
------------------------
|adr|len|com|d00|crh|crl|     setchar
------------------------
  1   2   3   4   5   6   byte

len = 1

reply sent by slave:
--------------------
|adr|len|sta|crh|crl|
--------------------
  1   2   3   4   5    byte

len = 0
```

The filter setting d00 may have the following values:

symbol	dec. value	hex value
OFF	120	0x78
FAST	121	0x79
MEDIUM	122	0x7A
SLOW	123	0x7B
SPECIAL	124	0x7C
BOX	125	0x7D

The parameter filter may have any of these values causing also the corresponding filtering effect after initializing the filter system.

Get the Web Temperature Filter Status:

I7GTLPF 101 0x65

```
command sent by master:
----------------------
|adr|len|com|crh|crl|   getchar
----------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
------------------------
|adr|len|sta|d00|crh|crl|
------------------------
  1   2   3   4   5   6   byte

len = 1
```

The data may be one of the following:

symbol	dec. value	hex value
on, data is filtered	1	0x01
off, no filter	0	0x00

Refer to User's Manual for more details. Using the filter will slow down the response time but will lower the noise. This filter does not affect the EXTRA thermometer input signal in any way nor does it affect the head temperature signal.

Set the Web Temperature Filter:

I7STLPF 99 0x63

```
command sent by master:
-------------------------
|adr|len|com|d00|crh|crl|     setchar
-------------------------
  1   2   3   4   5   6   byte

len = 1

reply sent by slave:
---------------------
|adr|len|sta|crh|crl|
---------------------
  1   2   3   4   5    byte

len = 0
```

The d00 field is used as follows:

mode may have any of the following values:

OFF	0	0x00	web temperature filter is off
ON	1	0x01	web temperature filter is on

Get the Locking Status:
I7GETLOCK 53 0x35

```
command sent by master:
----------------------
|adr|len|com|crh|crl|   getchar
----------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
------------------------
|adr|len|sta|d00|crh|crl|
------------------------
  1   2   3   4   5   6   byte

len = 1
```

The data may be one of the following:

symbol	dec. value	hex value
on, locked	1	0x01
off, autoranging	0	0x00

Refer to User's Manual for more details. Do not lock the gain if you do not know the resulting effects!

Set the Locking:
I7GAINLOCK 51 0x33

```
command sent by master:
---------------------
|adr|len|com|crh|crl|   setcom
---------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
---------------------
|adr|len|sta|crh|crl|
---------------------
  1   2   3   4   5      byte

len = 0
```

The command will change the locking accordingly and it will stay until set again, independent on power cutoff. After this command, the gain is locked. Using this command will avoid signal transients in critical measurements which could be resulted by the autoranging system changing automatically the system gain. Signal clipping and resulting strong distortion may occur if light signal increases too much. Refer to User's Manual for more details. Do not lock the gain if you do not know the resulting effects! This command has no effect in today's instruments and is obsolete.

Set the Autoranging:
I7GAINOPEN 52 0x34

```
command sent by master:
---------------------
|adr|len|com|crh|crl|   setcom
---------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
---------------------
|adr|len|sta|crh|crl|
---------------------
  1   2   3   4   5      byte

len = 0
```

The command will change the locking accordingly and it will stay until set again, independent on power cutoff. After this command, the gain is unlocked and the autoranging system is able to adapt to varying web conditions. Refer to **I7GAINLOCK** command.

Get the Lamp Status:
I7GLAMP 74 0x4A

```
command sent by master:
----------------------
|adr|len|com|crh|crl|   getchar
----------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
------------------------
|adr|len|sta|d00|crh|crl|
------------------------
  1   2   3   4   5   6   byte

len = 1
```

Data may be one of the following:

symbol	dec. value	hex value
OK	1	0x01
fault	0	0x00

Refer to notes in **I7GSTATUS**.

Get the Chopper Speed:
I7GFREQ 60 0x3C

```
command sent by master:
---------------------
|adr|len|com|crh|crl|   getfloat
---------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
-----------------------------------
|adr|len|sta|d00|d01|d02|d03|crh|crl|
-----------------------------------
  1   2   3   4   5   6   7   8   9    byte

len = 4
```

The received data bytes d00 - d03 are used for returning the light source speed in Herz. The form of the data is the following:

d00	high byte of frequency whole part as **int**
d01	low byte of frequency whole part as **int**
d02	high byte of frequency fract part as **int**
d03	low byte of frequency fract part as **int**

The actual frequency is calculated as follows:

fwhole = d00 * 256 + d01 (int)
ffract = d02 * 256 + d03 (int)
f (Hz) = fwhole + (ffract / 10000.0) (float)

Get the Usage Counter Hours:
I7GETUSG 28 0x1C

```
command sent by master:
---------------------
|adr|len|com|crh|crl|   getfloat
---------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
-----------------------------------
|adr|len|sta|d00|d01|d02|d03|crh|crl|
-----------------------------------
  1   2   3   4   5   6   7   8   9    byte

len = 4
```

The received data bytes d00 - d03 are used for returning the usage in hours (scaled down). The form of the data is the following:

d00 high byte of whole part as **int**
d01 low byte of whole part as **int**
d02 high byte of fract part as **int**
d03 low byte of fract part as **int**

The actual item is calculated as follows:

whole = d00 * 256 + d01 (int)
fract = d02 * 256 + d03 (int)
usage (h) = (whole + (fract / 10000.0)) * 1000.0 (float)

Set the Terminal Mode ON (Keyboard Mode):

I7STERM 47 0x2F

```
command sent by master:
---------------------
|adr|len|com|crh|crl|   setcom
---------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
---------------------
|adr|len|sta|crh|crl|
---------------------
  1   2   3   4   5      byte

len = 0
```

As a result of this command, the Keyboard mode is turned on in the RS232 serial port interface.

Get the Low Power Mode Status:
I7GETLPM 37 0x25

```
command sent by master:
----------------------
|adr|len|com|crh|crl|   getchar
----------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
------------------------
|adr|len|sta|d00|crh|crl|
------------------------
  1   2   3   4   5   6   byte

len = 1
```

Data may have one of the following values:

symbol	dec. value	hex value
Low Power	1	0x01
Normal	0	0x00

Set the Low Power mode ON/OFF, (ON = Low Power Mode, OFF = Normal Mode)
I7SETLPM 38 0x26

```
command sent by master:
------------------------
|adr|len|com|d00|crh|crl|     setchar
------------------------
  1   2   3   4   5   6   byte

len = 1

reply sent by slave:
---------------------
|adr|len|sta|crh|crl|
---------------------
  1   2   3   4   5    byte

len = 0
```

The d00 field is used as follows and the instrument's LowPower mode is set accordingly.

mode may have any of the following values:

OFF	0	0x00
ON	1	0x01

Get the Voltage Output Source:
I7GVOUT 88 0x58

```
command sent by master:
----------------------
|adr|len|com|crh|crl|   getchar
----------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
------------------------
|adr|len|sta|d00|crh|crl|
------------------------
  1   2   3   4   5   6    byte

len = 1
```

The selection may be one of the following:

symbol	dec. value	hex value
Moisture	0	0x00
Web temperature	1	0x01
Head temperature	2	0x02
Extra temperature	3	0x03

Set the Voltage Output Source:
I7SVOUT 87 0x57

```
command sent by master:
-------------------------
|adr|len|com|d00|crh|crl|     setchar
-------------------------
  1   2   3   4   5   6   byte

len = 1

reply sent by slave:
---------------------
|adr|len|sta|crh|crl|
---------------------
  1   2   3   4   5    byte

len = 0
```

d00 output field is used as follows:

symbol	dec. value	hex. value
Moisture	0	0x00
Web temperature	1	0x01
Head temperature	2	0x02
Extra temperature	3	0x03

The parameter will change the selection with a delay of 5 ms maximum to be seen at the analog output connector.

Get the Cooler Enable Status:
I7GCOOLING 90 0x5A

```
command sent by master:
----------------------
|adr|len|com|crh|crl|   getchar
----------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
------------------------
|adr|len|sta|d00|crh|crl|
------------------------
  1   2   3   4   5   6  byte

len = 1
```

The status may be one of the following:

symbol	dec. value	hex value
on	1	0x01
off	0	0x00

Refer to User's Manual for more details. Do not enable the cooler unless you know the resulting effects! To use the cooler, it must be installed into your meter.

Get the Cooler Temperature:
I7GCOOLTMP 93 0x5D

```
command sent by master:
----------------------
|adr|len|com|crh|crl|   getfloat
----------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
-----------------------------------
|adr|len|sta|d00|d01|d02|d03|crh|crl|
-----------------------------------
  1   2   3   4   5   6   7   8   9    byte

len = 4
```

The received data bytes d00 - d03 are used for returning the cooler temperature in C. The form of the data is the following:

d00 high byte of temp whole part as **int**
d01 low byte of temp whole part as **int**
d02 high byte of temp fract part as **int**
d03 low byte of temp fract part as **int**

The actual temperature is calculated as follows:

Twhole = d00 * 256 + d01 (int)
Tfract = d02 * 256 + d03 (int)
T (C) = Twhole + (Tfract / 10000.0) (float)

The temperature retrieval is available only if your system **specifically** supports it. Refer to User's Manual for more details. Do not enable the cooler unless you know the resulting effects! To use the cooler, it must be installed in your meter.

Get the Cooler On/off Status:

I7GCOOLON 94 0x5E

```
command sent by master:
---------------------
|adr|len|com|crh|crl|   getchar
---------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
------------------------
|adr|len|sta|d00|crh|crl|
------------------------
  1   2   3   4   5   6  byte

len = 1
```

The data may be one of the following:

symbol	dec. value	hex value
on	1	0x01
off	0	0x00

Refer to User's Manual for more details. Do not enable the cooler unless you know the resulting effects! To use the cooler, it must be installed into your meter.

Get the Cooler Linking Status:
I7GCOOLINK 95 0x5F

```
command sent by master:
----------------------
|adr|len|com|crh|crl|   getchar
----------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
------------------------
|adr|len|sta|d00|crh|crl|
------------------------
  1   2   3   4   5   6  byte

len = 1
```

The data may be one of the following:

symbol	dec. value	hex value
on, linked to Low Power	1	0x01
off, not linked	0	0x00

Refer to User's Manual for more details. Do not enable the cooler unless you know the resulting effects! To use the cooler, it must be installed into your meter.

Get the Cooler Status:
I7GCOOLSTA 97 0x61

```
command sent by master:
----------------------
|adr|len|com|crh|crl|   getchar
----------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
------------------------
|adr|len|sta|d00|crh|crl|
------------------------
  1   2   3   4   5   6   byte

len = 1
```

The data may be one of the following:

symbol	dec. value	hex value
OK	1	0x01
FAIL,need more purge air	0	0x00

Refer to User's Manual for more details. Do not enable the cooler unless you know the resulting effects! To use the cooler, it must be installed into your meter.

Set the Cooling Enable:
I7SCOOLING 92 0x5C

```
command sent by master:
------------------------
|adr|len|com|d00|crh|crl|     setchar
------------------------
  1   2   3   4   5   6   byte

len = 1

reply sent by slave:
--------------------
|adr|len|sta|crh|crl|
--------------------
  1   2   3   4   5    byte

len = 0
```

The d00 field is used as follows:

symbol	dec. value	hex. value
cooler disabled	0	0x00
cooler enabled	1	0x01

Refer to User's Manual for more details. Do not enable the cooler unless you know the resulting effects! To use the cooler, it must be installed into your meter.

Set the Cooling Linking:
I7SCOOLINK 96 0x60

```
command sent by master:
-------------------------
|adr|len|com|d00|crh|crl|     setchar
-------------------------
  1   2   3   4   5   6   byte

len = 1

reply sent by slave:
---------------------
|adr|len|sta|crh|crl|
---------------------
  1   2   3   4   5    byte

len = 0
```

The d00 field is used as follows:

symbol	dec. value	hex. value
cooler not linked	0	0x00
cooler linked to Low Power	1	0x01

Refer to User's Manual for more details. Do not enable the cooler unless you know the resulting effects! To use the cooler, it must be installed in your meter.

Get the Web Temperature Offset:
I7GWEBB 103 0x67

```
command sent by master:
---------------------
|adr|len|com|crh|crl|   getfloat
---------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
-----------------------------------
|adr|len|sta|d00|d01|d02|d03|crh|crl|
-----------------------------------
  1   2   3   4   5   6   7   8   9    byte

len = 4
```

The received data bytes d00 - d03 are used for returning the web temperature offset in C. The form of the data is the following:

d00	high byte of whole part as **int**
d01	low byte of twhole part as **int**
d02	high byte of fract part as **int**
d03	low byte of tfract part as **int**

The actual item is calculated as follows:

Twhole = d00 * 256 + d01 (int)
Tfract = d02 * 256 + d03 (int)
T (C) = Twhole + (Tfract / 10000.0) (float)

The offset is used for adjusting the thermometer in case it shows any long-term drift. The correct way to check it is to measure a temperature of an object of 21 - 26 C with a reliable meter and AK50's own web thermometer. The offset is then adjusted to match the reading of the reference meter. Do not do this at any other temperature as it may lead to increased nonlinearity of the signal! There may already be some reading for a unit leaving the factory. That is quite normal.

Set the Offset for Web Temperature:
I7SWEBB 102 0x66

```
command sent by master:
-----------------------------------
|adr|len|com|d00|d01|d02|d03|crh|crl|     setfloat
-----------------------------------
  1   2   3   4   5   6   7   8   9   byte

len = 4

reply sent by slave:
--------------------
|adr|len|sta|crh|crl|
--------------------
  1   2   3   4   5      byte

len = 0
```

The sent data bytes d00 - d03 are used for replacing the web temperature offset in C. The form of the data is the following:

d00 high byte of whole part as **int**
d01 low byte of twhole part as **int**
d02 high byte of fract part as **int**
d03 low byte of tfract part as **int**

The actual item would be calculated as follows:

Twhole = d00 * 256 + d01 (int)
Tfract = d02 * 256 + d03 (int)
T (C) = Twhole + (Tfract / 10000.0) (float)

With this command one can force the result of the web temperature to change if there is any reason to expect the thermometer's reading to have some sort of long-term drift.

Get the Head Overtemp Status:
I7GALM 104 0x68

```
command sent by master:
----------------------
|adr|len|com|crh|crl|   getchar
----------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
------------------------
|adr|len|sta|d00|crh|crl|
------------------------
  1   2   3   4   5   6   byte

len = 1
```

Data may be one of the following:

symbol	dec. value	hex value
OK, no alarm	0	0x00
overheating of the head	1	0x01

Refer to **I7CALM** for more details.

Clear the Head Overheating Alarm:

I7CALM 105 0x69

```
command sent by master:
---------------------
|adr|len|com|crh|crl|   setcom
---------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
---------------------
|adr|len|sta|crh|crl|
---------------------
  1   2   3   4   5      byte

len = 0
```

This command will clear the alarm situation which will persist **until** it is specifically cleared. Manual clearing is done in the Keyboard mode by entering and leaving the Menu at least once. Via Packet protocol the only way to clear the alarm is to use this command. Pay attention to the fact that the meter is put into Low Power mode as well. To restart it you have to restore it into Normal operation with the proper command. One should always find out the reasons for overheating to protect your investment and avoid any damage to the instrument. Use air purge and the optional internal cooler to avoid overheating. If that is not sufficient as your conditions for measurement are so extremely difficult, use some kind of heat shield to protect the meter. Consult Visilab if you need advice.

Calibration and Standardization Commands

Get the Current Material Entry:
I7GETMAT 14 0x0E

```
command sent by master:
---------------------
|adr|len|com|crh|crl|   getchar
---------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
------------------------
|adr|len|sta|d00|crh|crl|
------------------------
  1   2   3   4   5   6   byte

len = 1
```

As a response to this request, the current material entry - 1 is returned (entry = 1...100 or 0x00...0x64).

Switch to another Calibration Table in the Library:
I7SETMAT 15 0x0F

```
command sent by master:
------------------------
|adr|len|com|d00|crh|crl|     setchar
------------------------
  1   2   3   4   5   6   byte

len = 1

reply sent by slave:
--------------------
|adr|len|sta|crh|crl|
--------------------
  1   2   3   4   5    byte

len = 0
```

The d00 field is used as table entry (1...100) or (0x01...0x64).

The effect of using this command is switching to using another table in the calibration library. The switching takes actually some time and the resulting moisture values are not reliable for about two seconds. However, due to the nature of the situation of the table switching, this should not be no problem.

Get the Calibration Mode of the Current Material Entry:
I7GMODE 16 0x10

```
command sent by master:
---------------------
|adr|len|com|crh|crl|   getchar
---------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
------------------------
|adr|len|sta|d00|crh|crl|
------------------------
  1   2   3   4   5   6   byte

len = 1
```

Mode may be one of the following (the mode which is used for the current material entry):

symbol	dec. value	hex value
QUICK	78	0x4E
MULTI	79	0x4F

Remember that the QUICK mode is the global two-point calibration overriding the table chosen.

Set the Calibration Mode (MULTI/QUICK):

I7SMODE 17 0x11

```
command sent by master:
-------------------------
|adr|len|com|d00|crh|crl|     setchar
-------------------------
  1   2   3   4   5   6   byte

len = 1

reply sent by slave:
---------------------
|adr|len|sta|crh|crl|
---------------------
  1   2   3   4   5    byte

len = 0
```

The d00 field is used as follows:

symbol	dec. value	hex value
QUICK	78	0x4E
MULTI	79	0x4F

Read the Calibration Table Entry:
I7RXMAT 27 0x1B

```
command sent by master:
------------------------
|adr|len|com|d00|crh|crl|     special complex command
------------------------
  1   2   3   4   5   6    byte

len = 1

reply sent by slave:
-------------------------//--------------
|adr|len|sta|c00|c01|c02| // |c103|crh|crl|
-------------------------//--------------
  1   2   3   4   5   6       107  108 109   byte

len = 104
```

The data byte d00 is used for selecting the calibration entry (1...100). It does not have to be the current entry. All of the entry data is downloaded in one packet. The following format is used always independent on any other settings and data values.

c00: entry number of table to be read

c01: high byte of signal(0) whole part as **int**
c02: low byte of signal(0) whole part as **int**
c03: high byte of signal(0) fract part as **int**
c04: low byte of signal(0) fract part as **int**

c05: high byte of signal(1) whole part as **int**
c06: low byte of signal(1) whole part as **int**
c07: high byte of signal(1) fract part as **int**
c08: low byte of signal(1) fract part as **int**

c09: high byte of signal(2) whole part as **int**
c10: low byte of signal(2) whole part as **int**
c11: high byte of signal(2) fract part as **int**
c12: low byte of signal(2) fract part as **int**

c13: high byte of signal(3) whole part as **int**
c14: low byte of signal(3) whole part as **int**
c15: high byte of signal(3) fract part as **int**
c16: low byte of signal(3) fract part as **int**

c17: high byte of signal(4) whole part as **int**
c18: low byte of signal(4) whole part as **int**
c19: high byte of signal(4) fract part as **int**
c20: low byte of signal(4) fract part as **int**

c21: high byte of signal(5) whole part as **int**
c22: low byte of signal(5) whole part as **int**
c23: high byte of signal(5) fract part as **int**
c24: low byte of signal(5) fract part as **int**

c25: high byte of signal(6) whole part as **int**
c26: low byte of signal(6) whole part as **int**
c27: high byte of signal(6) fract part as **int**
c28: low byte of signal(6) fract part as **int**

c29: high byte of signal(7) whole part as **int**
c30: low byte of signal(7) whole part as **int**
c31: high byte of signal(7) fract part as **int**
c32: low byte of signal(7) fract part as **int**

c33: high byte of signal(8) whole part as **int**
c34: low byte of signal(8) whole part as **int**
c35: high byte of signal(8) fract part as **int**
c36: low byte of signal(8) fract part as **int**

c37: high byte of signal(9) whole part as **int**
c38: low byte of signal(9) whole part as **int**
c39: high byte of signal(9) fract part as **int**
c40: low byte of signal(9) fract part as **int**

c41: high byte of moisture(0) whole part as **int**
c42: low byte of moisture(0) whole part as **int**
c43: high byte of moisture(0) fract part as **int**
c44: low byte of moisture(0) fract part as **int**

c45: high byte of moisture(1) whole part as **int**
c46: low byte of moisture(1) whole part as **int**
c47: high byte of moisture(1) fract part as **int**
c48: low byte of moisture(1) fract part as **int**

c49: high byte of moisture(2) whole part as **int**
c50: low byte of moisture(2) whole part as **int**
c51: high byte of moisture(2) fract part as **int**
c52: low byte of moisture(2) fract part as **int**

c53: high byte of moisture(3) whole part as **int**
c54: low byte of moisture(3) whole part as **int**
c55: high byte of moisture(3) fract part as **int**
c56: low byte of moisture(3) fract part as **int**

c57: high byte of moisture(4) whole part as **int**
c58: low byte of moisture(4) whole part as **int**
c59: high byte of moisture(4) fract part as **int**
c60: low byte of moisture(4) fract part as **int**

c61: high byte of moisture(5) whole part as **int**
c62: low byte of moisture(5) whole part as **int**
c63: high byte of moisture(5) fract part as **int**
c64: low byte of moisture(5) fract part as **int**

c65: high byte of moisture(6) whole part as **int**
c66: low byte of moisture(6) whole part as **int**
c67: high byte of moisture(6) fract part as **int**
c68: low byte of moisture(6) fract part as **int**

c69: high byte of moisture(7) whole part as **int**
c70: low byte of moisture(7) whole part as **int**
c71: high byte of moisture(7) fract part as **int**
c72: low byte of moisture(7) fract part as **int**

c73: high byte of moisture(8) whole part as **int**
c74: low byte of moisture(8) whole part as **int**
c75: high byte of moisture(8) fract part as **int**
c76: low byte of moisture(8) fract part as **int**

c77: high byte of moisture(9) whole part as **int**
c78: low byte of moisture(9) whole part as **int**
c79: high byte of moisture(9) fract part as **int**
c80: low byte of moisture(9) fract part as **int**

c81: material entry name char 0
c82: material entry name char 1
c83: material entry name char 2
c84: material entry name char 3
c85: material entry name char 4
c86: material entry name char 5
c87: material entry name char 6
c88: material entry name char 7
c89: material entry name char 8
c90: material entry name char 9

c91: material entry name char 10
c92: material entry name char 11
c93: material entry name char 12
c94: material entry name char 13
c95: material entry name char 14
c96: material entry name char 15
c97: material entry name char 16
c98: material entry name char 17
c99: material entry name char 18
c100: material entry name char 19
c101: material entry name end marker (0) char 20

c102: material entry mode (QUICK/MULTI)
c103: material entry number of points in use (2..10)

As a response to this request, the packet fields, bytes 0 to 103 are filled for returning all table elements of signal and moisture. Note that the entry's material name can be read with another command I7GMATNM. The signal float values have been scaled up with 100.0 and one has to scale down the signal readings with 0.0100 to restore the original level. This is done only to keep a sufficient number of significant numbers in data transmission. The mirroring command is I7TXMAT.

Set the Calibration Table Entry:
I7TXMAT 26 0x1A

```
command sent by master:
-------------------------//--------------
|adr|len|com|c00|c01|c02| // |c103|crh|crl|     special complex command
-------------------------//--------------
  1   2   3   4   5   6       107  108 109   byte

len = 104

reply sent by slave:
---------------------
|adr|len|sta|crh|crl|
---------------------
  1   2   3   4   5      byte

len = 0
```

The data byte c00 is used for selecting the target calibration entry (1...100). It does not have to be the current entry. All of the entry data is uploaded in one packet. The following format is used always independent on any other settings and data values.

c00: entry number of table to be sent

c01: high byte of signal(0) whole part as **int**
c02: low byte of signal(0) whole part as **int**
c03: high byte of signal(0) fract part as **int**
c04: low byte of signal(0) fract part as **int**

c05: high byte of signal(1) whole part as **int**
c06: low byte of signal(1) whole part as **int**
c07: high byte of signal(1) fract part as **int**
c08: low byte of signal(1) fract part as **int**

c09: high byte of signal(2) whole part as **int**
c10: low byte of signal(2) whole part as **int**
c11: high byte of signal(2) fract part as **int**
c12: low byte of signal(2) fract part as **int**

c13: high byte of signal(3) whole part as **int**
c14: low byte of signal(3) whole part as **int**
c15: high byte of signal(3) fract part as **int**
c16: low byte of signal(3) fract part as **int**

c17: high byte of signal(4) whole part as **int**
c18: low byte of signal(4) whole part as **int**
c19: high byte of signal(4) fract part as **int**
c20: low byte of signal(4) fract part as **int**

c21: high byte of signal(5) whole part as **int**
c22: low byte of signal(5) whole part as **int**
c23: high byte of signal(5) fract part as **int**
c24: low byte of signal(5) fract part as **int**

c25: high byte of signal(6) whole part as **int**
c26: low byte of signal(6) whole part as **int**
c27: high byte of signal(6) fract part as **int**
c28: low byte of signal(6) fract part as **int**

c29: high byte of signal(7) whole part as **int**
c30: low byte of signal(7) whole part as **int**
c31: high byte of signal(7) fract part as **int**
c32: low byte of signal(7) fract part as **int**

c33: high byte of signal(8) whole part as **int**
c34: low byte of signal(8) whole part as **int**
c35: high byte of signal(8) fract part as **int**
c36: low byte of signal(8) fract part as **int**

c37: high byte of signal(9) whole part as **int**
c38: low byte of signal(9) whole part as **int**
c39: high byte of signal(9) fract part as **int**
c40: low byte of signal(9) fract part as **int**

c41: high byte of moisture(0) whole part as **int**
c42: low byte of moisture(0) whole part as **int**
c43: high byte of moisture(0) fract part as **int**
c44: low byte of moisture(0) fract part as **int**

c45: high byte of moisture(1) whole part as **int**
c46: low byte of moisture(1) whole part as **int**
c47: high byte of moisture(1) fract part as **int**
c48: low byte of moisture(1) fract part as **int**

c49: high byte of moisture(2) whole part as **int**
c50: low byte of moisture(2) whole part as **int**
c51: high byte of moisture(2) fract part as **int**
c52: low byte of moisture(2) fract part as **int**

c53: high byte of moisture(3) whole part as **int**
c54: low byte of moisture(3) whole part as **int**
c55: high byte of moisture(3) fract part as **int**
c56: low byte of moisture(3) fract part as **int**

c57: high byte of moisture(4) whole part as **int**
c58: low byte of moisture(4) whole part as **int**
c59: high byte of moisture(4) fract part as **int**
c60: low byte of moisture(4) fract part as **int**

c61: high byte of moisture(5) whole part as **int**
c62: low byte of moisture(5) whole part as **int**
c63: high byte of moisture(5) fract part as **int**
c64: low byte of moisture(5) fract part as **int**

c65: high byte of moisture(6) whole part as **int**
c66: low byte of moisture(6) whole part as **int**
c67: high byte of moisture(6) fract part as **int**
c68: low byte of moisture(6) fract part as **int**

c69: high byte of moisture(7) whole part as **int**
c70: low byte of moisture(7) whole part as **int**
c71: high byte of moisture(7) fract part as **int**
c72: low byte of moisture(7) fract part as **int**

c73: high byte of moisture(8) whole part as **int**
c74: low byte of moisture(8) whole part as **int**
c75: high byte of moisture(8) fract part as **int**
c76: low byte of moisture(8) fract part as **int**

c77: high byte of moisture(9) whole part as **int**
c78: low byte of moisture(9) whole part as **int**
c79: high byte of moisture(9) fract part as **int**
c80: low byte of moisture(9) fract part as **int**

c81: material entry name char 0
c82: material entry name char 1
c83: material entry name char 2
c84: material entry name char 3
c85: material entry name char 4
c86: material entry name char 5
c87: material entry name char 6
c88: material entry name char 7
c89: material entry name char 8
c90: material entry name char 9

c91: material entry name char 10
c92: material entry name char 11
c93: material entry name char 12
c94: material entry name char 13
c95: material entry name char 14
c96: material entry name char 15
c97: material entry name char 16
c98: material entry name char 17
c99: material entry name char 18
c100: material entry name char 19
c101: material entry name end marker (0) char 20

c102: material entry mode (QUICK/MULTI)
c103: material entry number of points in use (2..10)

The signal float values must be scaled up with 100.0. This is to keep a sufficient number of significant numbers in data transmission. The mirroring command is I7RXMAT.

Set the Offset for Standardization:
I7SSHIFT 67 0x43

```
command sent by master:
-----------------------------------
|adr|len|com|d00|d01|d02|d03|crh|crl|     setfloat
-----------------------------------
  1   2   3   4   5   6   7   8   9   byte

len = 4

reply sent by slave:
--------------------
|adr|len|sta|crh|crl|
--------------------
  1   2   3   4   5      byte

len = 0
```

The sent data bytes d00 - d03 are used for replacing the moisture or other signal offset belonging to the standardization system. The form of the data is the following:

d00 high byte of whole part as **int**
d01 low byte of twhole part as **int**
d02 high byte of fract part as **int**
d03 low byte of tfract part as **int**

The actual item would be calculated as follows:

swhole = d00 * 256 + d01 (int)
sfract = d02 * 256 + d03 (int)
s = swhole + (sfract / 10000.0) (float)

With this command one can force the result of the standardization to change (= "manual" standardization).

Set the Standard Moisture Value for Standardization:
I7SSTD 72 0x48

```
command sent by master:
-------------------------------------
|adr|len|com|d00|d01|d02|d03|crh|crl|     setfloat
-------------------------------------
  1   2   3   4   5   6   7   8   9   byte

len = 4

reply sent by slave:
---------------------
|adr|len|sta|crh|crl|
---------------------
  1   2   3   4   5      byte

len = 0
```

The sent data bytes d00 - d03 are used for replacing the moisture or other signal offset as measured from the standard. The form of the data is the following:

d00	high byte of whole part as **int**
d01	low byte of twhole part as **int**
d02	high byte of fract part as **int**
d03	low byte of tfract part as **int**

The actual item would be calculated as follows:

swhole = d00 * 256 + d01 (int)
sfract = d02 * 256 + d03 (int)
s = swhole + (sfract / 10000.0) (float)

With this command one can set the standard value of the constant moisture sample before performing the standardization. The value is kept in the meter's memory and this is not usually necessary to do more than once. If you change the moisture standard to another, you have to set this variable also.

Get the Material Entry Number Used in Standardization:
I7GSTDM 71 0x47

```
command sent by master:
----------------------
|adr|len|com|crh|crl|   getchar
----------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
------------------------
|adr|len|sta|d00|crh|crl|
------------------------
  1   2   3   4   5   6   byte

len = 1
```

Data returns the entry number (table entry 1...100).

Get the Offset Value Resulting from Standardization:
I7GSHIFT 68 0x44

```
command sent by master:
----------------------
|adr|len|com|crh|crl|   getfloat
----------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
-----------------------------------
|adr|len|sta|d00|d01|d02|d03|crh|crl|
-----------------------------------
  1   2   3   4   5   6   7   8   9    byte

len = 4
```

The received data bytes d00 - d03 are used for returning the offset in C. The form of the data is the following:

d00	high byte of whole part as **int**
d01	low byte of twhole part as **int**
d02	high byte of fract part as **int**
d03	low byte of tfract part as **int**

The actual item is calculated as follows:

ofswhole = d00 * 256 + d01 (int)
ofsfract = d02 * 256 + d03 (int)
ofs(%) = ofswhole + (ofsfract / 10000.0) (float)

Get the Standard Value Set for Standardization:
I7GSTD 73 0x49

```
command sent by master:
---------------------
|adr|len|com|crh|crl|   getfloat
---------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
-----------------------------------
|adr|len|sta|d00|d01|d02|d03|crh|crl|
-----------------------------------
  1   2   3   4   5   6   7   8   9    byte

len = 4
```

The received data bytes d00 - d03 are used for returning the actual standard sample reading in moisture percents. The form of the data is the following:

d00 high byte of whole part as **int**
d01 low byte of twhole part as **int**
d02 high byte of fract part as **int**
d03 low byte of tfract part as **int**

The actual item is calculated as follows:

whole = d00 * 256 + d01 (int)
fract = d02 * 256 + d03 (int)
std(%) = whole + (fract / 10000.0) (float)

Standardize:

I7STDZE 69 0x45

```
command sent by master:
----------------------
|adr|len|com|crh|crl|   setcom
----------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
----------------------
|adr|len|sta|crh|crl|
----------------------
  1   2   3   4   5      byte

len = 0
```

Note that the standardization operation may take about one minute. Refer to User's Manual for details.

Set the Standard Material Entry Number:
I7SSTDM 70 0x46

```
command sent by master:
-------------------------
|adr|len|com|d00|crh|crl|     setchar
-------------------------
  1   2   3   4   5   6   byte

len = 1

reply sent by slave:
---------------------
|adr|len|sta|crh|crl|
---------------------
  1   2   3   4   5    byte

len = 0
```

The d00 field is used as follows, **entry** (1...100) (0x01...0x64).

This entry number is used in the standardization operation.

Data Acquisition Commands

Get the Moisture:
I7MOIST 11 0x0B

```
command sent by master:
---------------------
|adr|len|com|crh|crl|   getfloat
---------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
-----------------------------------
|adr|len|sta|d00|d01|d02|d03|crh|crl|
-----------------------------------
  1   2   3   4   5   6   7   8   9    byte

len = 4
```

The received data bytes d00 - d03 are used for returning the measured moisture or other primary signal sent by the slave. The form of the data is the following:

d00 high byte of whole part as **int**
d01 low byte of twhole part as **int**
d02 high byte of fract part as **int**
d03 low byte of tfract part as **int**

The actual item is calculated as follows:

Swhole = d00 * 256 + d01 (int)
Sfract = d02 * 256 + d03 (int)
S = Swhole + (Sfract / 10000.0) (float)

Get the Optional Extra Web Temperature:
I7GWEB2 100 0x64

```
command sent by master:
----------------------
|adr|len|com|crh|crl|   getfloat
----------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
-----------------------------------
|adr|len|sta|d00|d01|d02|d03|crh|crl|
-----------------------------------
  1   2   3   4   5   6   7   8   9    byte

len = 4
```

The received data bytes d00 - d03 are used for returning the extra web temperature in C. The form of the data is the following:

d00 high byte of whole part as **int**
d01 low byte of twhole part as **int**
d02 high byte of fract part as **int**
d03 low byte of tfract part as **int**

The actual item is calculated as follows:

Twhole = d00 * 256 + d01 (int)
Tfract = d02 * 256 + d03 (int)
T (C) = Twhole + (Tfract / 10000.0) (float)

This is an optional feature in some meters having two web thermometers.

Get the Head Temperature:
I7GETTMP 46 0x2E

```
command sent by master:
---------------------
|adr|len|com|crh|crl|   getfloat
---------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
-----------------------------------
|adr|len|sta|d00|d01|d02|d03|crh|crl|
-----------------------------------
  1   2   3   4   5   6   7   8   9    byte

len = 4
```

The received data bytes d00 - d03 are used for returning the head temperature in C. The form of the data is the following:

d00 high byte of whole part as **int**
d01 low byte of twhole part as **int**
d02 high byte of fract part as **int**
d03 low byte of tfract part as **int**

The actual item is calculated as follows:

Twhole = d00 * 256 + d01 (int)
Tfract = d02 * 256 + d03 (int)
T (C) = Twhole + (Tfract / 10000.0) (float)

This command will return the head temperature as the data. However, it does NOT turn ON sending head temperature via DP polling the temperature field as defined with DP commands.

Get the Web Temperature:
I7GWEB 48 0x30

```
command sent by master:
---------------------
|adr|len|com|crh|crl|   getfloat
---------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
-----------------------------------
|adr|len|sta|d00|d01|d02|d03|crh|crl|
-----------------------------------
  1   2   3   4   5   6   7   8   9    byte

len = 4
```

The received data bytes d00 - d03 are used for returning the web temperature in C. The form of the data is the following:

d00 high byte of whole part as **int**
d01 low byte of twhole part as **int**
d02 high byte of fract part as **int**
d03 low byte of tfract part as **int**

The actual item is calculated as follows:

Twhole = d00 * 256 + d01 (int)
Tfract = d02 * 256 + d03 (int)
T (C) = Twhole + (Tfract / 10000.0) (float)

This command will return the web temperature as the data. However, it does NOT turn ON sending web temperature via DP polling the temperature field as defined with DP commands.

Get the Expansion Module Signal:
I7GXMOD 108 0x6C

```
command sent by master:
---------------------
|adr|len|com|crh|crl|   getfloat
---------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
-----------------------------------
|adr|len|sta|d00|d01|d02|d03|crh|crl|
-----------------------------------
  1   2   3   4   5   6   7   8   9    byte

len = 4
```

The received data bytes d00 - d03 are used for returning the expansion module signal. The form of the data is the following:

d00 high byte of whole part as **int**
d01 low byte of twhole part as **int**
d02 high byte of fract part as **int**
d03 low byte of tfract part as **int**

The actual item is calculated as follows:

Xwhole = d00 * 256 + d01 (int)
Xfract = d02 * 256 + d03 (int)
X (G) = Twhole + (Xfract / 10000.0) (float)

This command will return the expansion module signal as the data. The unit is now marked as G as in general, the unit is not known. The signal may have negative values as well. The optional module may be a special measuring unit or it may have some other purpose.

Memory Bank Commands

Read the Number of Samples in the Current Bank:
I7GETDM 35 0x23

```
command sent by master:
---------------------
|adr|len|com|crh|crl|   getfloat
---------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
-----------------------------------
|adr|len|sta|d00|d01|d02|d03|crh|crl|
-----------------------------------
  1   2   3   4   5   6   7   8   9    byte

len = 4
```

The received data bytes d00 - d03 are used for returning the number of samples in the current bank. The form of the data is the following:

d00	high byte of whole part as **int**
d01	low byte of twhole part as **int**
d02	high byte of fract part as **int**
d03	low byte of tfract part as **int**

The actual item is calculated as follows:

Nwhole = d00 * 256 + d01 (int)
Nfract = d02 * 256 + d03 (int)
N = Nwhole + (Nfract / 10000.0) (float)

This command will return the number of samples as the data (fract = 0).

Read the Bank Number:
I7GBANK 55 0x37

```
command sent by master:
---------------------
|adr|len|com|crh|crl|   getchar
---------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
------------------------
|adr|len|sta|d00|crh|crl|
------------------------
  1   2   3   4   5   6   byte

len = 1
```

Data returns the identifier of the current bank. It may be one of the following:

symbol	dec. value	hex. value
Series with 4096 points	0	0x00
Bank1 with 1024 points	1	0x01
Bank2 with 1024 points	2	0x02
Bank3 with 1024 points	3	0x03
Bank4 with 1024 points	4	0x04

Select the Bank:
I7SBANK 54 0x36

```
command sent by master:
------------------------
|adr|len|com|d00|crh|crl|     setchar
------------------------
  1   2   3   4   5   6   byte

len = 1

reply sent by slave:
--------------------
|adr|len|sta|crh|crl|
--------------------
  1   2   3   4   5    byte

len = 0
```

The d00 field is used as follows:

bank may be one of the following:

symbol	dec. value	hex. value
Series with 4096 points	0	0x00
Bank1 with 1024 points	1	0x01
Bank2 with 1024 points	2	0x02
Bank3 with 1024 points	3	0x03
Bank4 with 1024 points	4	0x04

The parameter **bank** causes immediate bank switching to the corresponding bank. Invalid bank references cause no action. Always one of the banks is selected as the current bank.

Get the Autotimer Status:

I7GETAUTO 43 0x2B

```
command sent by master:
----------------------
|adr|len|com|crh|crl|   getchar
----------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
------------------------
|adr|len|sta|d00|crh|crl|
------------------------
  1   2   3   4   5   6   byte

len = 1
```

Return the autotimer status. The status may be one of the following:

symbol	dec. value	hex value
ON	**1**	**0x01**
OFF	**0**	**0x00**

Clear the Current Data Series (or Bank):

I7CLRSER 21 0x15

```
command sent by master:
----------------------
|adr|len|com|crh|crl|   setcom
----------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
----------------------
|adr|len|sta|crh|crl|
----------------------
  1   2   3   4   5      byte

len = 0
```

This command will clear the data series in the current memory bank.

Take a Sample into the Current Data Series (or Bank):
I7SAMPLE 36 0x24

```
command sent by master:
----------------------
|adr|len|com|crh|crl|   setcom
----------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
----------------------
|adr|len|sta|crh|crl|
----------------------
  1   2   3   4   5      byte

len = 0
```

This command will add the latest moisture reading to the data series in the current bank.

Set the Autotimer ON:

I7AUTOON 41 0x29

```
command sent by master:
---------------------
|adr|len|com|crh|crl|   setcom
---------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
---------------------
|adr|len|sta|crh|crl|
---------------------
  1   2   3   4   5      byte

len = 0
```

This command will turn on the autotimer. It will stay on until it is turned off (continuous or Normal operation) or it will be turned off automatically (Batch mode) after fetching the predetermined number of samples. Even in Batch mode the autotimer can be turned off. This will affect the web temperature autotimer as well if they are linked together. Note that the web temperature data is collected to a special memory bank with a rate of 16 samples / s. In earlier SW versions, this rate was one sample / second. The Autotimer is used in the Burst mode too but its control is fully given to the mode as soon as the Autotimer is started. The preset number, Burst count, is acquired at preset time intervals of the Autotimer. Then, the Autotimer is turned off by the mode itself.

Set the Autotimer OFF:

I7AUTOOFF 42 0x2A

```
command sent by master:
---------------------
|adr|len|com|crh|crl|   setcom
---------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
---------------------
|adr|len|sta|crh|crl|
---------------------
  1   2   3   4   5      byte

len = 0
```

This command will turn off the autotimer, independent on the mode. This will affect the web temperature autotimer as well if they are linked together. The Autotimer is used in the Burst mode too but its control is fully given to the mode as soon as the Autotimer is started. The preset number, Burst count, is acquired at preset time intervals of the Autotimer. Then, the Autotimer is turned off by the mode itself.

Set the Autotimer Mode:
I7SAMODE 58 0x3A

```
command sent by master:
-------------------------
|adr|len|com|d00|crh|crl|     setchar
-------------------------
  1   2   3   4   5   6   byte

len = 1

reply sent by slave:
---------------------
|adr|len|sta|crh|crl|
---------------------
  1   2   3   4   5    byte

len = 0
```

The d00 field is used as follows:

mode symbol	dec. value	hex. value
Normal	1	0x01
Batch	0	0x00

The parameter will change the autotimer mode accordingly. The Burst mode has no use for this mode as it has its own "Batch" count, the Burst count.

Get the Autotimer Mode:
I7GAMODE 59 0x3B

```
command sent by master:
----------------------
|adr|len|com|crh|crl|   getchar
----------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
------------------------
|adr|len|sta|d00|crh|crl|
------------------------
  1   2   3   4   5   6   byte

len = 1
```

Return the autotimer mode. The mode may be one of the following:

symbol	dec. value	hex value
Normal	1	0x01
Batch	0	0x00

Get the Autotimer Interval in Seconds:
I7GETTIM 40 0x28

```
command sent by master:
---------------------
|adr|len|com|crh|crl|   getfloat
---------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
-----------------------------------
|adr|len|sta|d00|d01|d02|d03|crh|crl|
-----------------------------------
  1   2   3   4   5   6   7   8   9    byte

len = 4
```

The received data bytes d00 - d03 are used for returning the Autotimer sampling interval in (s). The form of the data is the following:

d00	high byte of interval whole part as **int**
d01	low byte of interval whole part as **int**
d02	high byte of interval fract part as **int**
d03	low byte of interval fract part as **int**

The actual interval is calculated as follows:

twhole = d00 * 256 + d01 (int)
tfract = d02 * 256 + d03 (int)
t (s) = twhole + (tfract / 10000.0) (float)

Set the Autotimer Interval in Seconds:
I7SETTIM 39 0x27

```
command sent by master:
-----------------------------------
|adr|len|com|d00|d01|d02|d03|crh|crl|     setfloat
-----------------------------------
  1   2   3   4   5   6   7   8   9   byte

len = 4

reply sent by slave:
--------------------
|adr|len|sta|crh|crl|
--------------------
  1   2   3   4   5      byte

len = 0
```

The sent data bytes d00 - d03 are used for replacing the Autotimer interval in (s). The form of the data is the following:

d00	high byte of whole part as **int**
d01	low byte of twhole part as **int**
d02	high byte of fract part as **int**
d03	low byte of tfract part as **int**

The actual item would be calculated as follows:

twhole = d00 * 256 + d01 (int)
tfract = d02 * 256 + d03 (int)
tint = twhole + (tfract / 10000.0) (float)

The setting may be from 0.0025 to 32000 s in increments of 0.0025 s. Invalid settings are limited in value before use in the slave.

Get the Current Batch Size:
I7GBATCH 57 0x39

```
command sent by master:
---------------------
|adr|len|com|crh|crl|   getfloat
---------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
-----------------------------------
|adr|len|sta|d00|d01|d02|d03|crh|crl|
-----------------------------------
  1   2   3   4   5   6   7   8   9    byte

len = 4
```

The received data bytes d00 - d03 are used for returning the number of samples in the Batch. The form of the data is the following:

d00 high byte of whole part as **int**
d01 low byte of twhole part as **int**
d02 high byte of fract part as **int**
d03 low byte of tfract part as **int**

The actual item is calculated as follows:

Nwhole = d00 * 256 + d01 (int)
Nfract = d02 * 256 + d03 (int)
N = Nwhole + (Nfract / 10000.0) (float)

This command will return the number of samples in Batch as the data (fract = 0).

Set the Current Batch Size:
I7SBATCH 56 0x38

```
command sent by master:
-----------------------------------
|adr|len|com|d00|d01|d02|d03|crh|crl|     setfloat
-----------------------------------
  1   2   3   4   5   6   7   8   9   byte

len = 4

reply sent by slave:
--------------------
|adr|len|sta|crh|crl|
--------------------
  1   2   3   4   5      byte

len = 0
```

The sent data bytes d00 - d03 are used for replacing the Batch size. The form of the data is the following:

d00	high byte of whole part as **int**
d01	low byte of twhole part as **int**
d02	high byte of fract part as **int**
d03	low byte of tfract part as **int**

The actual item would be calculated as follows:

bwhole = d00 * 256 + d01 (int)
bfract = d02 * 256 + d03 (int)
ba = bwhole + (bfract / 10000.0) (float)

The maximum of batch count is 32767. Exceeding values will be limited to this reading in all cases by the slave.

Get Samples from the Current Memory Bank:
I7TXSER 20 0x14

```
command sent by master:
------------------------
|adr|len|com|d00|crh|crl|     special complex command
------------------------
  1   2   3   4   5   6    byte

len = 1

reply sent by slave:
---------------------//--------------
|adr|len|sta|c01|c02| // |c103|crh|crl|
---------------------//--------------
  1   2   3   4   5       107  108 109   byte

len = 64
```

The data byte d00 is used for selecting the index of the first data in the bank array (1...4096). The number of points to be downloaded is always 16 independent on actual bank contents or number of samples actually saved into it. All 16 of the data elements are downloaded in one packet. The following format is used always independent on any other settings and data values.

c01: high byte of signal(0) whole part as **int**
c02: low byte of signal(0) whole part as **int**
c03: high byte of signal(0) fract part as **int**
c04: low byte of signal(0) fract part as **int**

c05: high byte of signal(1) whole part as **int**
c06: low byte of signal(1) whole part as **int**
c07: high byte of signal(1) fract part as **int**
c08: low byte of signal(1) fract part as **int**

c09: high byte of signal(2) whole part as **int**
c10: low byte of signal(2) whole part as **int**
c11: high byte of signal(2) fract part as **int**
c12: low byte of signal(2) fract part as **int**

c13: high byte of signal(3) whole part as **int**
c14: low byte of signal(3) whole part as **int**
c15: high byte of signal(3) fract part as **int**
c16: low byte of signal(3) fract part as **int**

c17: high byte of signal(4) whole part as **int**
c18: low byte of signal(4) whole part as **int**
c19: high byte of signal(4) fract part as **int**
c20: low byte of signal(4) fract part as **int**

c21: high byte of signal(5) whole part as **int**
c22: low byte of signal(5) whole part as **int**
c23: high byte of signal(5) fract part as **int**
c24: low byte of signal(5) fract part as **int**

c25: high byte of signal(6) whole part as **int**
c26: low byte of signal(6) whole part as **int**
c27: high byte of signal(6) fract part as **int**
c28: low byte of signal(6) fract part as **int**

c29: high byte of signal(7) whole part as **int**
c30: low byte of signal(7) whole part as **int**
c31: high byte of signal(7) fract part as **int**
c32: low byte of signal(7) fract part as **int**

c33: high byte of signal(8) whole part as **int**
c34: low byte of signal(8) whole part as **int**
c35: high byte of signal(8) fract part as **int**
c36: low byte of signal(8) fract part as **int**

c37: high byte of signal(9) whole part as **int**
c38: low byte of signal(9) whole part as **int**
c39: high byte of signal(9) fract part as **int**
c40: low byte of signal(9) fract part as **int**

c41: high byte of moisture(0) whole part as **int**
c42: low byte of moisture(0) whole part as **int**
c43: high byte of moisture(0) fract part as **int**
c44: low byte of moisture(0) fract part as **int**

c45: high byte of moisture(1) whole part as **int**
c46: low byte of moisture(1) whole part as **int**
c47: high byte of moisture(1) fract part as **int**
c48: low byte of moisture(1) fract part as **int**

c49: high byte of moisture(2) whole part as **int**
c50: low byte of moisture(2) whole part as **int**
c51: high byte of moisture(2) fract part as **int**
c52: low byte of moisture(2) fract part as **int**

c53: high byte of moisture(3) whole part as **int**
c54: low byte of moisture(3) whole part as **int**
c55: high byte of moisture(3) fract part as **int**
c56: low byte of moisture(3) fract part as **int**

c57: high byte of moisture(4) whole part as **int**
c58: low byte of moisture(4) whole part as **int**
c59: high byte of moisture(4) fract part as **int**
c60: low byte of moisture(4) fract part as **int**

c61: high byte of moisture(5) whole part as **int**
c62: low byte of moisture(5) whole part as **int**
c63: high byte of moisture(5) fract part as **int**
c64: low byte of moisture(5) fract part as **int**

As a response to this request, the packet fields, bytes c01 to c64 are filled for returning all data elements of samples in the bank. The values have NOT been scaled up since usually there is no danger of losing significant numbers.

The actual moisture is calculated as follows for each of the 16 samples:

moisture (%) = moisture whole + moisture fractional / 100

Note that the index to be used depends on the bank you are using. Also the number of samples actually collected into it is important. If you have taken only 128 samples but are indexing at sample number 256 to start, the data will be totally improper (garbage in, garbage out). Also, for example, if you index to 255 and have taken only 256 samples into the array, ignore samples after #256. If bank boundaries are exceeded, the data may be zero or invalid.

Copy the Temperature Series to Bank4:
I7COPYT 98 0x62

```
command sent by master:
----------------------
|adr|len|com|crh|crl|   setcom
----------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
----------------------
|adr|len|sta|crh|crl|
----------------------
  1   2   3   4   5      byte

len = 0
```

This command copies the data series collected with the temperature autotimer to an independent bank to the globally visible Bank4. The original special memory bank is cleared.

Get the Current Burst Size:
I7GBURST 113 0x71

```
command sent by master:
---------------------
|adr|len|com|crh|crl|   getfloat
---------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
-----------------------------------
|adr|len|sta|d00|d01|d02|d03|crh|crl|
-----------------------------------
  1   2   3   4   5   6   7   8   9    byte

len = 4
```

The received data bytes d00 - d03 are used for returning the current burst size. The form of the data is the following:

d00	high byte of whole part as **int**
d01	low byte of twhole part as **int**
d02	high byte of fract part as **int**
d03	low byte of tfract part as **int**

The actual item is calculated as follows:

Nwhole = d00 * 256 + d01 (int)
Nfract = d02 * 256 + d03 (int)
N = Nwhole + (Nfract / 10000.0) (float)

This command will return the burst size as the data (fract = 0).

Set the Current Burst Size:
I7SBURST 112 0x70

```
command sent by master:
-----------------------------------
|adr|len|com|d00|d01|d02|d03|crh|crl|     setfloat
-----------------------------------
  1   2   3   4   5   6   7   8   9   byte

len = 4

reply sent by slave:
--------------------
|adr|len|sta|crh|crl|
--------------------
  1   2   3   4   5      byte

len = 0
```

The sent data bytes d00 - d03 are used for replacing the burst size. The form of the data is the following:

d00	high byte of whole part as **int**
d01	low byte of twhole part as **int**
d02	high byte of fract part as **int**
d03	low byte of tfract part as **int**

The actual item would be calculated as follows:

bwhole = d00 * 256 + d01 (int)
bfract = d02 * 256 + d03 (int)
bz = bwhole + (bfract / 10000.0) (float)

The maximum allowed burst size is 32767.

Set the Burst Mode:
I7SBUM 114 0x72

```
command sent by master:
------------------------
|adr|len|com|d00|crh|crl|     setchar
------------------------
  1   2   3   4   5   6   byte

len = 1

reply sent by slave:
--------------------
|adr|len|sta|crh|crl|
--------------------
  1   2   3   4   5    byte

len = 0
```

The d00 field is used as follows:

symbol	dec. value	hex. value
burst mode off	0	0x00
burst mode on	1	0x01

Refer to User's Manual for more details of the burst mode. It overrides the normal Autotimer operation in some ways.

Get the Burst Mode:
I7GBUM 115 0x73

```
command sent by master:
---------------------
|adr|len|com|crh|crl|   getchar
---------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
------------------------
|adr|len|sta|d00|crh|crl|
------------------------
  1   2   3   4   5   6   byte

len = 1
```

The data may be one of the following:

symbol	dec. value	hex value
burst mode on	1	0x01
burst mode off	0	0x00

Refer to User's Manual for more details.

Get the Current Burst Mode Item Count:
I7GBUC 116 0x74

```
command sent by master:
---------------------
|adr|len|com|crh|crl|   getfloat
---------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
-----------------------------------
|adr|len|sta|d00|d01|d02|d03|crh|crl|
-----------------------------------
  1   2   3   4   5   6   7   8   9    byte

len = 4
```

The received data bytes d00 - d03 are used for returning the number of items (scaled down) accumulated while using the burst mode. The form of the data is the following:

d00 high byte of whole part as **int**
d01 low byte of twhole part as **int**
d02 high byte of fract part as **int**
d03 low byte of tfract part as **int**

The actual item is calculated as follows:

Nwhole = d00 * 256 + d01 (int)
Nfract = d02 * 256 + d03 (int)
N = (Nwhole + (Nfract / 10000.0)) * 100.0 (float)

This command will return the number of items as the data.

Clear the Current Burst Mode Item Count:
I7CBUC 117 0x75

```
command sent by master:
----------------------
|adr|len|com|crh|crl|   setcom
----------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
----------------------
|adr|len|sta|crh|crl|
----------------------
  1   2   3   4   5      byte

len = 0
```

This command will clear the item counter used with the burst mode.

Text String Commands

Get the Unit for Moisture:
I7GUNIT 13 0x0D

```
command sent by master:
---------------------
|adr|len|com|crh|crl|   getchar
---------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
-------------------------------------------
|adr|len|sta|c01|c02|c03|c04|c05|c06|crh|crl|
-------------------------------------------
  1   2   3   4   5   6   7   8   9   10  11   byte

len = variable, up to 6
```

The response data consists entirely of character data forming the reply string of unit for the measuring system (like %, V, Ohm, pF etc.). The string length is **len**. The bytes are as follows.

c01: first byte of string as **char**
c02: second byte of string as **char**
...

There is not necessarily an end marker in the string so one has to add it to the end for safety. The string is of variable length and describes the user-editable measuring unit. The maximum length of the unit is 6 characters including the end marker.

Get the Current Material Entry Name:
I7GMATNM 31 0x1F

```
command sent by master:
---------------------
|adr|len|com|crh|crl|   getchar
---------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
----------------------//--------------
|adr|len|sta|c01|c02| // |c21|crh|crl|
----------------------//--------------
  1   2   3   4   5       24  25 26   byte

len = variable, up to 21
```

The response data consists entirely of character data forming the reply string. The string length is **len**. The bytes are as follows.

c01: first byte of string as **char**
c02: second byte of string as **char**
...

There is not necessarily an end marker in the string so one has to add it to the end for safety. The string is of varible length and describes the calibration table name given by the user. Note that the name may contain spaces and special characters too. The maximum length of the name is 21 characters including the end marker.

Get the Current Library Name:
I7GLIBNM 29 0x1D

```
command sent by master:
---------------------
|adr|len|com|crh|crl|   getchar
---------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
-----------------------------------------------------------
|adr|len|sta|c01|c02|c03|c04|c05|c06|c07|c08|c09|crh|crl|
-----------------------------------------------------------
  1   2   3   4   5   6   7   8   9   10  11  12  13  14 byte

len = variable, up to 9
```

The response data consists entirely of character data forming the reply string. The string length is **len**. The bytes are as follows.

c01: first byte of string as **char**
c02: second byte of string as **char**
...

There is not necessarily an end marker in the string so one has to add it to the end for safety. The string is of varible length and describes the calibration library name given by the user. Note that the name may contain spaces and special characters too. The maximum length of the name is maximum 9 characters long including the end marker. It is not recommended to have spaces and special characters in the name since many PC applications wish to use this string as a suggestion for a file name.

Get the Meter's Identifier String:

I7TEST 10 0x0A

```
command sent by master:
---------------------
|adr|len|com|crh|crl|   getchar
---------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
----------------------//--------------
|adr|len|sta|c01|c02| // |c122|crh|crl|
----------------------//--------------
  1   2   3   4   5       125  126 127   byte

len = variable, up to 122
```

The response data consists entirely of character data forming the reply string. The string length is **len**. The bytes are as follows.

c01: first byte of string as **char**
c02: second byte of string as **char**
...

There is not necessarily an end marker in the string so one has to add it to the end for safety. The string is of varible length and describes the main identifiers of the instrument model, its serial number etc all concatenated into the same string with space separators.

Get the Expansion Module Name:
I7GXNAME 110 0x6E

```
command sent by master:
---------------------
|adr|len|com|crh|crl|   getchar
---------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
-----------------------------------------------------------
|adr|len|sta|c01|c02|c03|c04|c05|c06|c07|c08|c09|crh|crl|
-----------------------------------------------------------
  1   2   3   4   5   6   7   8   9   10  11  12  13  14   byte

len = variable, up to 8
```

The response data consists entirely of character data forming the reply string. The string length is **len**. The bytes are as follows.

c01: first byte of string as **char**
c02: second byte of string as **char**
...

There is not necessarily an end marker in the string so one has to add it to the end for safety. The string is of varible length and describes the name given by the manufacturer to the expansion module. The maximum length of the name is maximum 9 characters long including the end marker. The name is factory set when the module is installed to the meter. It's existence is shown in status3 also. The name is a short description of what the module does. It may be a special measuring unit or it may have some other purpose.

Set the Current Library Name:
I7SLIBNM 30 0x1E

```
command sent by master:
-----------------------------------------------------
|adr|len|com|c01|c02|c03|c04|c05|c06|c07|c08|c09|crh|crl|
-----------------------------------------------------
  1   2   3   4   5   6   7   8   9   10  11  12  13  14   byte

len = variable, up to 9. This has to reflect on the actual packet length too!

reply sent by slave:
---------------------
|adr|len|sta|crh|crl|   getchar
---------------------
  1   2   3   4   5      byte

len = 0
```

The set data consists entirely of character data forming the string. The string length is **len**. The bytes are as follows.

c01: first byte of string as **char**
c02: second byte of string as **char**
...

Use an end marker in the string. The string is of variable length and describes the calibration library name given by the user. The maximum length of the name is maximum 9 characters long including the end marker. It is not recommended to have spaces and special characters in the name since many PC applications wish to use this string as a suggestion for a file name. The end marker may be at any position indicating the actual length of the string. In that case, the rest of the data will be ignored. You can fill in the rest of the output bytes with zero. Remember that **len** belongs to the packet protocol and if it is set to an incorrect value, the packet will be ignored.

Set the Unit for Moisture:
I7SUNIT 12 0x0C

```
command sent by master:
-----------------------------------------------
|adr|len|com|c01|c02|c03|c04|c05|c06|c07|crh|crl|
-----------------------------------------------
  1   2   3   4   5   6   7   8   9   10  11  12    byte

len = variable, up to 7. This has to reflect on the actual packet length too!

reply sent by slave:
---------------------
|adr|len|sta|crh|crl|   getchar
---------------------
  1   2   3   4   5      byte

len = 0
```

The set data consists entirely of character data forming the string. The string length is **len**. The bytes are as follows.

c01: first byte of string as **char**
c02: second byte of string as **char**
...

Use an end marker in the string. The string is of variable length and describes the measuring unit given by the user. The maximum length of the unit is maximum 7 characters long including the end marker. It is not recommended to have spaces and special characters in the name. The end marker may be at any position indicating the actual length of the string. In that case, the rest of the data will be ignored. You can fill in the rest of the output bytes with zero. Remember that **len** belongs to the packet protocol and if it is set to an incorrect value, the packet will be ignored. The unit is maximum 6 characters long. Note that the unit does not affect the resulting moisture value in any way. It is displayed in Keyboard mode only in the PC program's screen. It is also inquired at various instances while saving data files.

Special Commands

Initialize the Profibus DP slave:
I7DPINIT 65 0x41

```
command sent by master:
---------------------
|adr|len|com|crh|crl|   setcom
---------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
---------------------
|adr|len|sta|crh|crl|
---------------------
  1   2   3   4   5      byte

len = 0
```

This will initialize the DP slave. Note that it will also be incapable of responding to DP master while initializing all SW and HW subsystems (< 1 second).

Send a Short Pulse to the LED Indicator (if available on the connector panel):
I7BEEP 34 0x22

```
command sent by master:
----------------------
|adr|len|com|crh|crl|   setcom
----------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
----------------------
|adr|len|sta|crh|crl|
----------------------
  1   2   3   4   5      byte

len = 0
```

This command will flash once the optional indicator lamp on the back panel of the meter.

Start Fast Fourier Transform in the Meter:
I7FFT 83 0x53

```
command sent by master:
---------------------
|adr|len|com|crh|crl|   setcom
---------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
---------------------
|adr|len|sta|crh|crl|
---------------------
  1   2   3   4   5      byte

len = 0
```

The meter's own FFT routine is started with this command. It is a fixed 1024 samples' transform and it uses the contents of Bank1. While performing it, it needs the space of the other banks, Bank2, Bank3, and Bank4. The long Series is not touched during that process. The conversion will take usually less than ten seconds and the resulting power spectrum will be placed into Bank1. The spectrum is not flattened with logarithm, it is linear. One can use it if the **IRMA7Basic or Advanced** program is not available or there are no frequency domain algorithms available in your development tools. The data from Bank1 can then be downloaded to your PC for further analysis and display.

Get the Expansion Module Number:
I7GNXMOD 109 0x6D

```
command sent by master:
----------------------
|adr|len|com|crh|crl|   getchar
----------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
------------------------
|adr|len|sta|d00|crh|crl|
------------------------
  1   2   3   4   5   6   byte

len = 1
```

Return the number of the expansion module connected to this meter. The number (0...255) is factory set when the module is installed to the meter. It's existence is shown in status3 also. The number is a condensed description of the type of the module and gives the acquisition system instructions on how to handle the associated data.The module may be a special measuring unit or it may have some other purpose.

Send a Command to the Expansion Module:
I7SXCOM 111 0x6F

```
command sent by master:
-------------------------
|adr|len|com|d00|crh|crl|     setchar
-------------------------
  1   2   3   4   5   6   byte

len = 1

reply sent by slave:
---------------------
|adr|len|sta|crh|crl|
---------------------
  1   2   3   4   5    byte

len = 0
```

The d00 field is used as follows:

symbol	dec. value	hex. value
command	0...255	0x00...0xFF

The **command** will be directly transferred to the expansion module without any filtering, interpretation or waiting for a reply. There is only a simple protocol associated with this, no actual error checking. The module may be a special measuring unit or it may have some other purpose. If error checking or command accepting response is required by the expansion module, it will respond by the signal it transmits. That is interpreted at the highest level to make sure the command was accepted without errors. Refer to the expansion module's own User's Guide for details about its commands, if any are available. If no expansion module is installed to **AK50** or another instrument, the commands have no effect.

Get the DP Address:

I7GDPADR 61 0x3D

```
command sent by master:
---------------------
|adr|len|com|crh|crl|   getchar
---------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
------------------------
|adr|len|sta|d00|crh|crl|
------------------------
  1   2   3   4   5   6   byte

len = 1
```

Return the address of the Profibus DP slave connected to the DP fieldbus. The number (0...255) is both factory set when the meter is configured the first time and the user can decide it later. Also the DP master can change the address while running.

Get the Profibus DP Active Status:
I7GDPACT 66 0x42

```
command sent by master:
----------------------
|adr|len|com|crh|crl|   getchar
----------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
------------------------
|adr|len|sta|d00|crh|crl|
------------------------
  1   2   3   4   5   6   byte

len = 1
```

Return the active status of this DP slave (1=Active, 0 = inactive). The slave can be made inactive and the master will detect this. Activating/deactivating are made with corresponding commands.

Set the Profibus DP Active:
I7DPACT 63 0x3F

```
command sent by master:
----------------------
|adr|len|com|crh|crl|   getchar
----------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
------------------------
|adr|len|sta|d00|crh|crl|
------------------------
  1   2   3   4   5   6   byte

len = 1
```

Set the DP slave Active.

Set the Profibus DP Inactive:

I7DPDEACT 64 0x40

```
command sent by master:
----------------------
|adr|len|com|crh|crl|   getchar
----------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
------------------------
|adr|len|sta|d00|crh|crl|
------------------------
  1   2   3   4   5   6   byte

len = 1
```

Set the DP slave Inactive.

No Operation Command NOP:
I7NOP 91 0x5B

```
command sent by master:
----------------------
|adr|len|com|crh|crl|   getchar
----------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
------------------------
|adr|len|sta|d00|crh|crl|
------------------------
  1   2   3   4   5   6   byte

len = 1
```

This command has no effect as the slave refuses to do anything about it except reply to the command as usual. This command is good for protocol testing and message passing timing testing etc.

Set Profibus DP Slave Address:
I7SDPADR 62 0x3E

```
command sent by master:
------------------------
|adr|len|com|d00|crh|crl|     setchar
------------------------
  1   2   3   4   5   6   byte

len = 1

reply sent by slave:
--------------------
|adr|len|sta|crh|crl|
--------------------
  1   2   3   4   5    byte

len = 0
```

The d00 field is used as the DP slave address.

symbol	dec. value	hex. value
address	0...255	0x00...0xFF

The number (0...255) is both factory set when the meter is configured the first time and the user can decide it later. Also the DP master can change the address while running.

Get the Limit Switch High Level:
I7GETHI 32 0x20

```
command sent by master:
---------------------
|adr|len|com|crh|crl|   getfloat
---------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
-----------------------------------
|adr|len|sta|d00|d01|d02|d03|crh|crl|
-----------------------------------
  1   2   3   4   5   6   7   8   9    byte

len = 4
```

The received data bytes d00 - d03 are used for returning the upper limit. The form of the data is the following:

d00 high byte of whole part as **int**
d01 low byte of twhole part as **int**
d02 high byte of fract part as **int**
d03 low byte of tfract part as **int**

The actual item is calculated as follows:

Lwhole = d00 * 256 + d01 (int)
Lfract = d02 * 256 + d03 (int)
L (%) = Lwhole + (Lfract / 10000.0) (float)

With this command one can get the upper limit of the switch. When the limit value has been exceeded, the HI switch is closed, else it is open. This is an optional feature in some meters requiring isolated switches for external control.

Get the Limit Switch Low Level:
I7GETLO 33 0x21

```
command sent by master:
---------------------
|adr|len|com|crh|crl|   getfloat
---------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
-----------------------------------
|adr|len|sta|d00|d01|d02|d03|crh|crl|
-----------------------------------
  1   2   3   4   5   6   7   8   9    byte

len = 4
```

The received data bytes d00 - d03 are used for returning the lower limit. The form of the data is the following:

d00 high byte of whole part as **int**
d01 low byte of twhole part as **int**
d02 high byte of fract part as **int**
d03 low byte of tfract part as **int**

The actual item is calculated as follows:

Lwhole = d00 * 256 + d01 (int)
Lfract = d02 * 256 + d03 (int)
L (%) = Lwhole + (Lfract / 10000.0) (float)

With this command one can get the lower limit of the switch. When the signal has passed under the limit value, the LO switch is closed, else it is open. This is an optional feature in some meters requiring isolated switches for external control.

Set the Limit Switch High Level:
I7SETHI 18 0x12

```
command sent by master:
-----------------------------------
|adr|len|com|d00|d01|d02|d03|crh|crl|     setfloat
-----------------------------------
  1   2   3   4   5   6   7   8   9   byte

len = 4

reply sent by slave:
--------------------
|adr|len|sta|crh|crl|
--------------------
  1   2   3   4   5      byte

len = 0
```

The sent data bytes d00 - d03 are used for replacing the upper limit value. The form of the data is the following:

d00	high byte of whole part as **int**
d01	low byte of twhole part as **int**
d02	high byte of fract part as **int**
d03	low byte of tfract part as **int**

The actual item would be calculated as follows:

Lwhole = d00 * 256 + d01 (int)
Lfract = d02 * 256 + d03 (int)
L = Lwhole + (Lfract / 10000.0) (float)

With this command one can set the upper limit of the switch. When the limit value has been exceeded, the HI switch is closed, else it is open. This is an optional feature in some meters requiring isolated switches for external control.

Set the Limit Switch Low Level:
I7SETLO 19 0x13

```
command sent by master:
-----------------------------------
|adr|len|com|d00|d01|d02|d03|crh|crl|     setfloat
-----------------------------------
  1   2   3   4   5   6   7   8   9   byte

len = 4

reply sent by slave:
--------------------
|adr|len|sta|crh|crl|
--------------------
  1   2   3   4   5      byte

len = 0
```

The sent data bytes d00 - d03 are used for replacing the lower limit value. The form of the data is the following:

d00 high byte of whole part as **int**
d01 low byte of twhole part as **int**
d02 high byte of fract part as **int**
d03 low byte of tfract part as **int**

The actual item would be calculated as follows:

Lwhole = d00 * 256 + d01 (int)
Lfract = d02 * 256 + d03 (int)
L = Lwhole + (Lfract / 10000.0) (float)

With this command one can set the lower limit of the switch. When the signal passes under the limit value, the LO switch is closed, else it is open. This is an optional feature in some meters requiring isolated switches for external control.

Prepare Configuration Data Download:
I7CDPREP 106 0x6A

```
command sent by master:
----------------------
|adr|len|com|crh|crl|   setcom
----------------------
  1   2   3   4   5      byte

len = 0

reply sent by slave:
----------------------
|adr|len|sta|crh|crl|
----------------------
  1   2   3   4   5      byte

len = 0
```

This command forces the meter to prepare data into the current memory bank ready for transfer. The data consists of all important instrument configuration which is saved to some data archive for future use. Do not apply this command unless you are aware of all of its consequences. After a short delay, one can download the data array from the current memory bank for further saving. Restoring the data should be done with the aid of IRMA7Basic or Advanced programs to avoid any troubles with the instrument. We are not publishing details of the mirroring command I7CDSET.

Appendix 1. Schematic of Electrical Connections for Model D in the connection box. **If you open the distribution boxes cover, please, disconnect the power lead first to avoid a shock!** Find below a schematic for wiring of the circuit board inside the power supply -PS. On the next page, find a corresponding component placement drawing. You can find more information from the model -PS Operating guide.

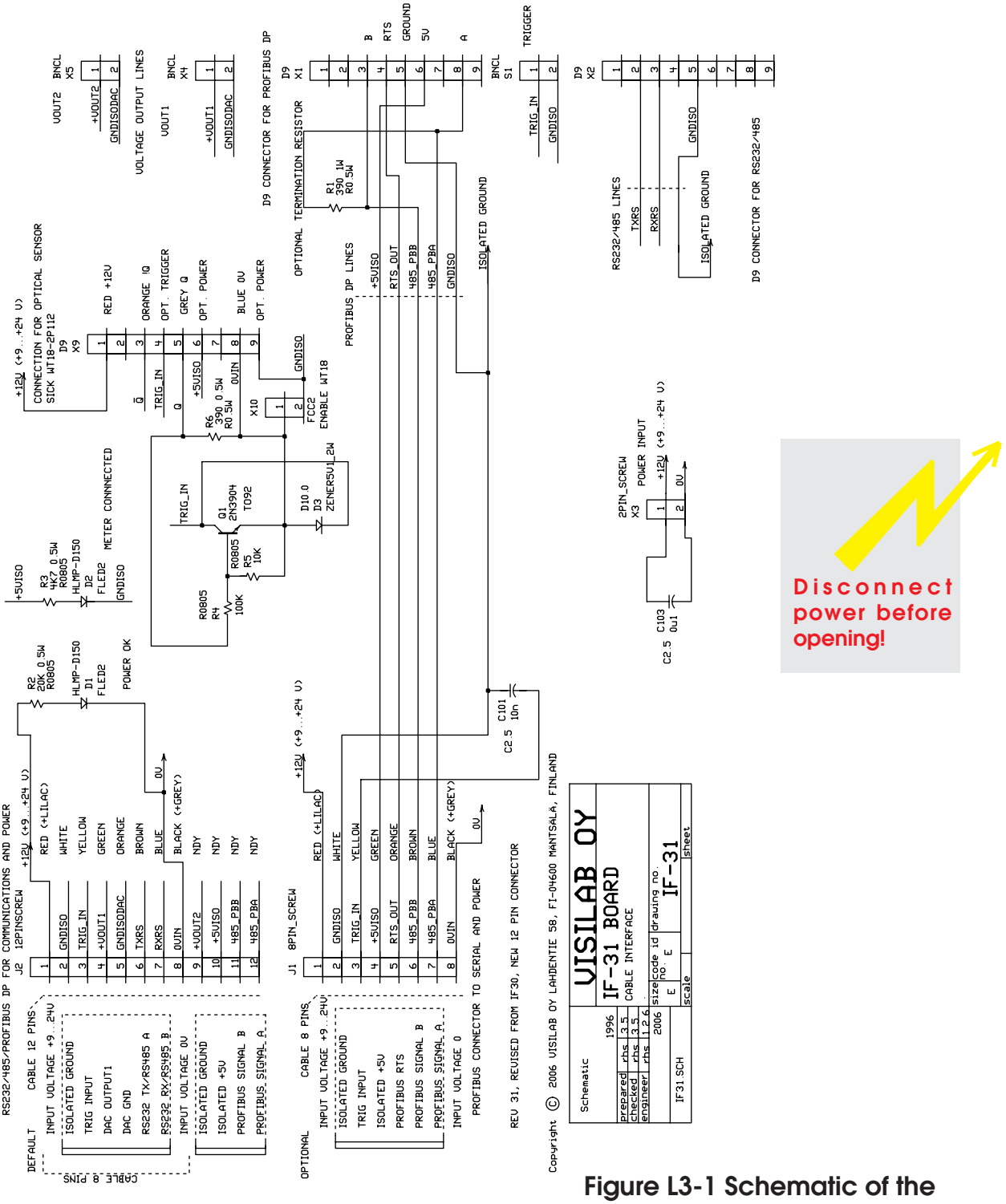


Figure L3-1 Schematic of the circuit board in the model -PS power supply unit

Appendix 2. Technical Specifications of the Protocol

The **AK50** moisture meters communicate with the external word by using a packet protocol containing error checking and full recovery from all error events.The main principle is a genuine master-slave action. There is only one master and several slaves (max 255). The protocol works as a half duplex system allowing only one node to talk at a time and only after a permission to do so. The master always is the initiating party and the slave is a passive responder. The protocol is not dependent on the lowest level hardware implementation but can operate over many practical hardware platforms. Usually only timing issues have to be checked to ensure proper working. One can use either RS232 with a transparent bridging unit, RS485 as a two-wire HDX, four-wire HDX or even Ethernet. The only requirement is that the delays are predictable which makes it a real-time system, at least in some time scale. The baud rates are not significant either. The protocol delays have to be adjusted accordingly.

For the protocol to work, there must not be two slaves with the same address. Else, there will be a mixture of colliding packets and everybody seems to be right in sending packets and ignoring the other party's packets. This means that no slave can have the master address either.

The basic sequence is that the master sends out a packet to a particular slave. The packet has a target address, a command and the size of thee data part (if any). Then there is the data part and at the end the CRC check counts. The CRC is calculated according to the standard CRCCCIT 0x1021 base.

The slave will identify the packet to be addressed to him and continues listening and saving the packet contents. Then it processes the contents and replies to it as required. Some commands require only a simple packet to be returned having the same structure (target address, status, length of data, CRC) but the data part does not exist (len=0). The rest of the job is at the slave to obey the command and complete the actions requested. Typically, there are commands for setting some variables or modes and for getting their values. Every command must always be replied to. If the save does not recognize the command for itself, it does not respond at all. The slave sends nothing by itself, ever. This principle guarantees that there are no collisions in the bus as long as the addresses are correct and there is no physical damage in any node.

The data part is variable in length and the length is required to be correct in every packet. The maximum size of the packet is 127 bytes and for the data part it is 122 bytes. Five bytes are always used for the frame overhead.

Basic Rules for the Slave

The slave must follow these rules:

1. The addresses are divided 0 = master, 1...255 = slave
2. the slaves have always unique addresses. it is not allowed for any party to change the address suddenly (unless the user wishes so)
3. the slave must not send anything himself, only as an immediate reply to a master's command
4. if there is any error whatsoever in receiving the packet, the packet must be rejected and nothing is replied. Resending the command is the job of the master.
5. The frame format is variable as to the packet size but otherwise it is fixed for speed and simplicity. Certain commands expect packets with a certain format as a reply. This makes programming easier and data transmission faster.
6. Error checking is based on slave address, packet size and CRC.
7. The maximum size of the packet is 127 bytes, the smallest is five bytes.
8. The only timing important for the slave is the intercharacter delay in receiving. If that is

exceeded, the packet is considered faulty.
9. The default length of the packet is five bytes and it is changed to a variable length as soon as it indicated in the frame.
10. If the slave received characters while processing and starting to send back its reply, it will stop transmission and reject the packet in hand. It will wait for the new command. however, it is possible that it has already done some actions according to the earlier command which seemed all right to it.

If the slave detects any error in the packet, it will wait for another command and must not reply to the command at all. It needs to listen to the end of the packet to determine correctly when a new proper packet is arriving.

Reasons for rejecting the packet are the following:

1. An incorrect CRC in the packet: Comparison between the arrived CRC vs. calculated CRC
2. A too long delay between characters
3. Incorrect address ==> listen to the end (not an error)
4. incorrect data part length
5. Too many bytes in the packet
6. Characters received while processing reply or replying ==> stop processing, error

The only confusing situation will appear when the data part is marked as too small bytes are still received after that. This is an error, of course. Detection of any of these errors will cause the slave to wait for a proper packet again. But that means that the master must itself detect the error too and possibly make a resending. This slows down data transmission. However, the speed is very good and errors happen very rarely if the protocol has been implemented correctly both i slaves and in the master and all delays are set properly. Observe carefully that the slave always needs some time to process with the packet and other actions to reply properly. it has also other duties, like measuring and analysis tasks at the same time and, depending on system, this may cause momentary overloading to the slave. The master delay should be set long enough to avoid any errors caused by an impatient master.

Basic Rules for the Master

The master must follow the rules:

1. The addresses are divided: 0 = master, 1...255 = slave
2. The master transacts with only one slave at a time. There are no global commands and no such command into which more than one slave are allowed to reply.
3. If someone is talking in the network without the permission of the master, that is an error.
4. If there is any error whatsoever in receiving the reply, that is interpreted as an error and causes resending the command.
5. The frame format is flexible as to the packet or data part length. Certain commands are replied with certain size replies for simplicity, however.
6. The error checking is according to the packet size and CRC count.
7. The maximum size of the packet is 127 bytes and the smallest is five bytes.
8. The number of resendings must be limited to avoid stalling the network. For some reason the number can be increased but must be lowered again.
9. The maser must take into account the processing and action times with some commands to avoid errors. E.g. if the command requests a complete reinitialization of the slave, it is useless to send another command immediately to that slave as its bootup sequence can take half a minute.
10. There are no global or synchronizing commands. If a group of slaves needs to have the

same command, each one has to receive it separately. For best timing accuracy, use hardware timing for synchronization.
11. The only delay which is important for the master is the maximum intercharacter delay, no other delays are required.
12. The sending of the packet bytes must be within the slave's reception delay, else the slave delay will be exceeded and the master will never get a reply. This is especially important with Windows-based masters since Windows is not a real-time system at all and may inadvertently buffer outgoing data and send it out when it pleases. That is not a bad thing itself but the sendout of the buffer must be done in sequence without any extra delays after that.
13. The master must be more patient the slave since slave can be doing signal processing or other time-consuming tasks while receiving the packet. It is not going to reply it endangering its own internal tasks and signal accuracy which may be important for the process. Take this into account in the architecture of the system.

If the master detects reception of a faulty packet, it can resend the command. Reasons for resending are:

1. Faulty CRC in the received packet
2. a too long delay between characters ==> resend. This and the conditions above handle also the case of too little number of bytes in the packet.
3. Incorrect target address (= 0) ==> listen to the end and resend the command
4. Unexpected data traffic in the networking the middle of sending a command==> interrupt the transmission, listen to the end and resend
5. too many bytes in the packet ==> resend

Resending should be done only a limited number of times after a specific error. Error status should be passed to the upper level if the maximum number of resendings has been exceeded as an indication of persisting data transmission problem with the slave or in the network. The usual reason is likely that the slave is offline.

The packet frame structure has been specified in Fig. 2-1. The first byte is the target address. It is 0 for the master and 1...255 for the slave. The second byte is the size of the data part and the third byte is the command to the slave or the status from the slave. The following bytes form the data and the last two bytes are the CRC count high and low bytes. The commands are defined separately and both parties must speak the same language and dialect. AK50 products try to use the same language as much as is reasonable and very few commands are model-specific. The C language is regularly used with unsigned int and unsigned char as variables in type-sensitive tasks, like CRC calculation.

When building a data acquisition system with several instruments, be they moisture meters or whatever, we can offer some advice and recommendations which facilitate programming and the final user as well.

1. The slaves are installed in the order of slave addresses or vice versa to understand their position in the process (1, 2, 3, ...N). If the meters have to be sent for service or are used elsewhere temporarily, the addresses have to be maintained consistently.

2. After bootup the master does not have to assume anything else of the system and slave ordering than what was said in the first paragraph, not even that. The Advanced software, for example, has a searching routine for looking up the list of available slaves indicating their models and serial numbers etc. This will facilitate building up the system after a break production. The same software is able to save the slave addresses properly to be remembered at the next launching without any further effort.

3. A single meter can be disconnected from the system without stalling the whole system. Before that, the master should be told that the slave is going to drop off.

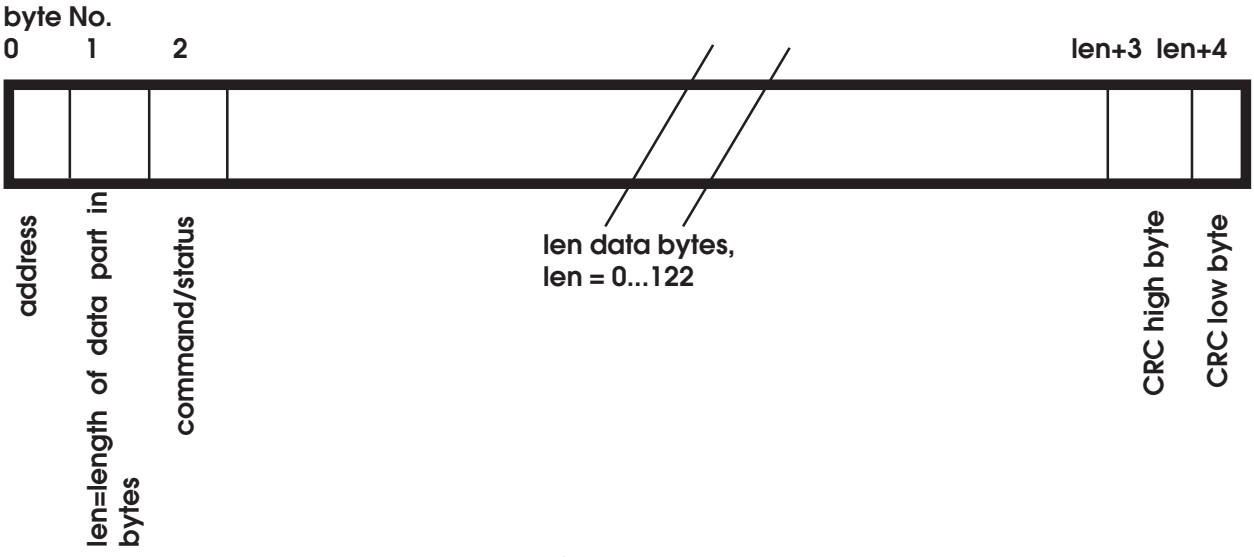
4. Fault diagnostics should be developed as far as possible to detect any anomalies in the system and to ensure acquisition of all important data. Reporting to user is recommended too. The master could be able to disconnect a faulty slave itself.

These are the problem which the slave is able to report itself:

1. an overheated meter
2. clipping of signals
3. web break or suspected web break
5. status bytes in each meter

The commands have internal structures with fractional numeric data or char/int data parts. Refer to the separate descriptions in this manual. Fixed-point real numbers are used for reading or setting some variable. Some variables are scaled up or down to preserve the number of significant numbers in data transmission.

Fig. 2-1. Frame of the packet.



Appendix 3. Examples of the CRC calculation

Definitions of CRC variables and functions:

```
#define CRCCCITT  0X1021  /* STANDARD POLYNOMIAL IN GENERAL USE! */
#define CRCCCITT_REV    0X8408      /* reverse CCITT polynomial */
#define CRC16           0X8005  /* CRC16 plynomial */
#define CRC16_REV 0XA001  /* reverse CRC16 plynomial */

#define crcupdate( d, a, t)    *(a) = (*(a) << 8) ^ (t)[(*(a) >> 8) ^ (d)];
#define crcrevupdate( d, a, t)  *(a) = ((*(a) >> 8) ^ (t)[(*(a) ^(d)) &
0X00FF]);

#define DATAPAK 130

#define MASTER 38
#define SLAVE  39
#define RESENDCOUNT 10
#define TIMEOUT 30      /* timeout in milliseconds */

/* positions of the control bytes in the packet */
#define TAGTPOS  0
#define LENPOS   1
#define COMPOS   2
#define MAXPOS   3  /* first character position after the packet header */

#define CRCHIPOS 3
#define CRCLOPOS 4

#define PAKOVHD 5   /* number of characters in addition to data */


#define MASTOUT  40000L       /* MASTER timeout */
#define SLAVOUT   3000L       /* SLAVE timeout */

#define MASERDLY  50          /* MASTER ERROR timeout */

#define RECOVERABLE    7 /* errors at the sending end */
#define UNRECOVERABLE  9
#define TIMEOUTERR    19
#define ADDRERR       20 /* slave uses an illegal master address */
#define TXCRCERROR    29 /* if the packet from master is not OK */
                  /* as replied by slave */

#define RXOK          78 /* no errors at the receiving end */

#define RXCRCERROR    66 /* if the response packet from slave is not OK */
                  /* as calculated by master. Also the slave may */
                  /* come to the conclusion that the CRC of the  */
                  /* master's packet is incorrect */
#define RXPAKERROR    88
#define RXFRAMEERROR 101 /* general error in packet frame or timeout */
#define RXNOMSG      102 /* no messages received */
```

```
#define RXUNKNOWN    103 /* unknown command *
```

Sample calls for functions in this project:

initialization of the CRC table:

```
  tablep = mk_crctbl( CRCCCITT, crchware);
```

calculation of CRC:

```
  stxCRC = calc_CRC( bufout, len);
```

```
/*+++++++++++++*/

unsigned int calc_CRC( char *buffer, char pak_len)
{
/*

  This function calculates the CRC.
  Always initialize the CRC table before using it, at least once.

  Use unsigned variables where expressly identified, else use signed
variables
*/
int  i;
unsigned int final;
char templo;
int c;

final = 0;
i = 0;
while( i < pak_len + MAXPOS)
   {
   templo = buffer[ i];
   c = (int)( templo);
   crcupdate( c, &final, tablep);
   i++;
   };

/* the CRC is in final */
return( final);
}

/*+++++++++++++*/

unsigned int crchware( unsigned int data, unsigned int genpoly,
   unsigned int accum)
```

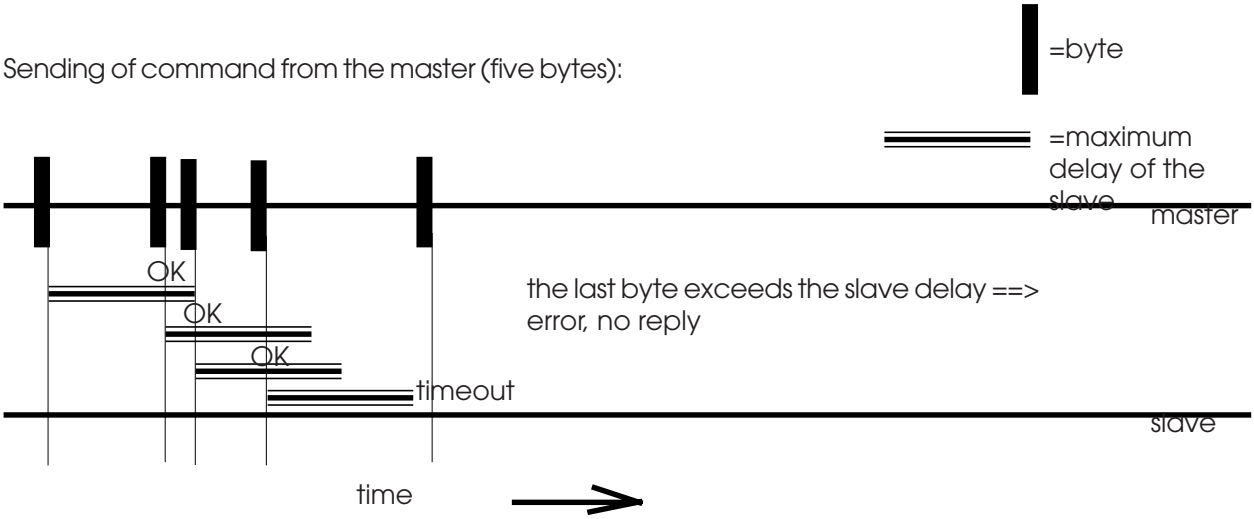
```
{
/*
  Simulates a CRC hardware.
  Generates a CRC directly. Produces the
  same remainder as polynomial division
  with 2 NULL bytes appended to the message.
*/
char i;

data <<= 8;

for( i = 8; i > 0; i—)
   {
   if(( data ^ accum) & 0X8000) /* if MSB of (data XOR accum) is TRUE */
      {
      accum = (accum << 1) ^ genpoly; /* then shift and subtract poly */
      }
   else
      {
      accum <<= 1;  /* otherwise transparent shift */
      };
   data <<= 1;  /* move up to next bit for XOR */
   };

return( accum);
}

/*+++++++++++++*/

unsigned int *mk_crctbl( unsigned int poly, unsigned int (*crcfn)())
/*i, pol, z)
   int i;
   unsigned int pol;
   int z;*/
{
/*
    Builds a CRC lookup table based
    upon the specified polynomial
    and CRC function. Each table element
    contains the CRC of its rank in the
    table.
*/
unsigned int *crctp;
int i;

crctp = crctbl;

for( i = 0; i < 256; i++)  /* fill table with CRC's of values 0..255 */
   crctp[ i] = (*crcfn)( (int)i, (unsigned int)poly, (int)0);

return( crctp);  /* return a ptr to table */
}

/*+++++++++++++++*/
```

Appendix 4. Delay Analysis for the Packet Protocol

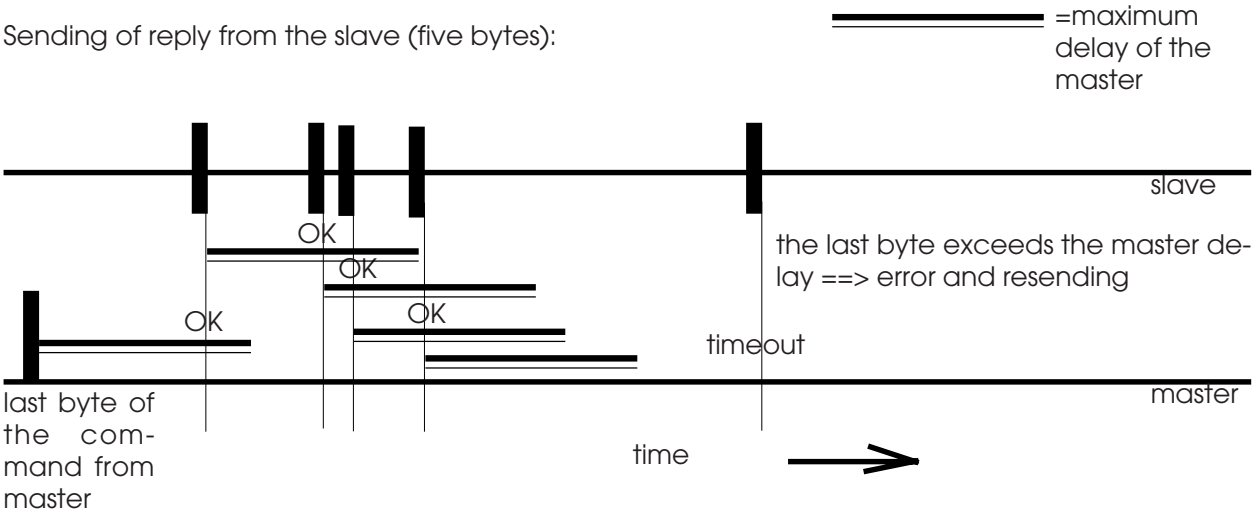
A reliable communication between master and slave is based on properly set timing, maximum time delays. Exceeding these delays will cause an error. Master has to determine the error since slave will not reply if it has detected some sort of time-out. The slave delays need to be shorter than master delays for the protocol to work. Fine-tuning the delays is done with the aid of testing to make sure that no errors are detected in normal operation. The delays should be as short as possible to ensure quick recovery from error. This is in contrast to the requirement of good detection of a genuine delay problem. This protocol denies all natural collisions when everything is working normally and thus is an optimal protocol having maximum reliability.

Collisions can happen only in cases of misprogramming, faulty nodes and incorrect addresses. Special situations may arise if tailored transparent bridges with long internal delays are used or some other inaccurate hardware bus is residing under the protocol. Faulty bytes are detected and

Sending of command from the master (five bytes):



Sending of reply from the slave (five bytes):



do not cause delay problems since the whole packet is rejected.

Refer to the drawings for relations between the delays. The generally used baud rate is 9600 and 34800 bauds. The lower rate allows approximately 250 bytes of effective data transmission speed. testing will reveal the resulting loading capability.

The delays are related as follows:

Tdelslave << Tdelmaster

The master must be more patient than the slave to keep up the protocol. The slave's delay can be rather short. A basic delay should be added to the master to allow the slave to handle each command properly.

Appendix 5. Numerical Values of Commands and Constants Used in the Packet Protocol

```
/*
      ____________________________________


       HEADER FILE FOR PACKET PROTOCOL


         Copyright (C) 1994 - 2009 Visilab Signal Technologies Oy

                  Finland

      ____________________________________


             FILE      : PACKET.H
             PURPOSE   : HEADER FILE FOR PACKET.C
             CREATED   : 10.7.1994 BY RHS/VISILAB
             MODIFIED  : 17.2.1997 commands connected  with levels
                            have been removed
                            23.7.99 restored them again

   Note that some commands are reserved and some commands are used in the Profibus DP
   only.

*/

#define DATAPAK 130

#define MASTER 38
#define SLAVE  39
#define RESENDCOUNT 10

/* commands in
            packet protocol         Profibus DP implementation */
#define I7TEST   10
#define I7MOIST  11
#define I7SUNIT  12
#define I7GUNIT  13
#define I7GETMAT 14
#define I7SETMAT 15                /* DP */
#define I7GMODE  16
#define I7SMODE  17                /* DP */
#define I7SETHI  18
#define I7SETLO  19
#define I7TXSER  20
#define I7CLSER  21                /* DP */
#define I7TXMAT  26
#define I7RXMAT  27
#define I7GETUSG 28
#define I7GLIBNM 29
#define I7SLIBNM 30
#define I7GMATNM 31
#define I7GETHI  32
#define I7GETLO  33
#define I7BEEP   34                /* DP */
```

```
#define I7GETDM  35
#define I7SAMPLE 36                /* DP */
#define I7GETLPM 37
#define I7SETLPM 38                /* DP */
#define I7SETTIM 39
#define I7GETTIM 40
#define I7AUTOON 41                /* DP */
#define I7AUTOOFF 42               /* DP */
#define I7GETAUTO 43
#define I7GRAWS   44              /* reserved, not in all models */
#define I7GRAWR   45              /* reserved, not in all models */
#define I7GETTMP  46              /* DP */

/* extra commands not in standard IRMA-7-A instrumentation : */
#define I7STERM   47              /* DP */
#define I7GWEB    48              /* DP */

#define I7SFILTER  49             /* DP */
#define I7GFILTER  50
#define I7GAINLOCK 51
#define I7GAINOPEN 52
#define I7GETLOCK  53
#define I7SBANK    54             /* DP */
#define I7GBANK    55
#define I7SBATCH   56
#define I7GBATCH   57
#define I7SAMODE   58             /* DP */
#define I7GAMODE   59
#define I7GFREQ    60
#define I7GDPADR   61
#define I7SDPADR   62             /* not available */
#define I7DPACT    63             /* not available */
#define I7DPDEACT  64             /* not available */
#define I7DPINIT   65             /* DP */
#define I7GDPACT   66
#define I7SSHIFT   67
#define I7GSHIFT   68
#define I7STDZE    69             /* DP */
#define I7SSTDM    70             /* DP */
#define I7GSTDM    71
#define I7SSTD     72
#define I7GSTD     73
#define I7GLAMP    74
#define I7SPACKET  75              /* DP */
#define I7GSTATUS  76              /* DP */
#define I7GMATNM2  77
#define I7TEST2    78
#define I7GHEAD    79 /* not LAN */   /* DP only */
#define I7TEST3    80
#define I7SMATNM   81
#define I7SMATNM2  82
#define I7FFT      83
#define I7GLAN     84 /* not LAN */
#define I7SLAN     85 /* not LAN */
#define I7G2STATUS 86              /* DP */
#define I7SVOUT    87
#define I7GVOUT    88
#define I7G3STATUS 89              /* DP ok */
```

```
#define I7GCOOLING 90              /* DP ok */
#define NOP        91 /* No Operation Command! */
#define I7SCOOLING 92               /* DP ok */
#define I7GCOOLTMP 93               /* DP ok */
#define I7GCOOLON  94               /* DP ok */
#define I7GCOOLINK 95               /* DP ok */
#define I7SCOOLINK 96               /* DP ok */
#define I7GCOOLSTA 97               /* DP ok */
#define I7COPYT    98               /* DP ok */
#define I7STLPF    99               /* DP ok */
#define I7GWEB2   100 /* a special command for an optional input */
#define I7GTLPF   101               /* DP ok */
#define I7SWEBB   102               /* DP ok */
#define I7GWEBB   103               /* DP ok */
#define I7GALM    104               /* DP ok */
#define I7CALM    105               /* DP ok */
#define I7CDPREP  106
#define I7CDSET   107
#define I7GXMOD   108               /* DP  */
#define I7GNXMOD  109               /* DP  */
#define I7GXNAME  110               /* DP  */
#define I7SXCOM   111               /* DP  */

/* new commands since V0.9BDP */
#define I7SBURST  112
#define I7GBURST  113
#define I7SBUM    114
#define I7GBUM    115
#define I7GBUC    116
#define I7CBUC    117
/*            = the last command */



/* filter selectors */
#define FILNONE   120
#define FILFAST   121
#define FILMEDM   122
#define FILSLOW   123
#define FILSPEC   124
#define FILBOX    125

#define RECOVERABLE    7 /* errors at the sending end */
#define UNRECOVERABLE  9
#define TIMEOUTERR    19
#define ADDRERR       20 /* slave uses an illegal master address */
#define TXCRCERROR    29 /* if the packet from master is not OK */
                        /* as replied by slave */

#define RXOK          78 /* no errors at the receiving end */

#define RXCRCERROR    66 /* if the response packet from slave is not OK */
                        /* as calculated by master. Also the slave may */
                        /* come to the conclusion that the CRC of the  */
                        /* master's packet is incorrect */
#define RXPAKERROR    88
#define RXFRAMEERROR 101 /* general error in packet frame or timeout */
```

```
#define RXNOMSG     102 /* no messages received */
#define RXUNKNOWN    103 /* unknown command */


/* platform dependent definitions */
#define TIMEOUT 30      /* timeout in milliseconds */

/* positions of the control bytes in the packet */
#define TAGTPOS  0
#define LENPOS   1
#define COMPOS   2
#define MAXPOS   3  /* first character position after the packet header */

#define CRCHIPOS 3
#define CRCLOPOS 4

#define PAKOVHD 5   /* number of characters in addition to data */


#define MASTOUT  40000L       /* MASTER timeout */
#define SLAVOUT   3000L       /* SLAVE timeout */

#define MASERDLY  50          /* MASTER ERROR timeout */
```

Index

A

analog output 12, 13
Autoranging 20
autotimer 68, 69
Autotimer Interval 72, 73
Autotimer Mode 71
autotimer mode 70
Autotimer Status 65

B

bank 64, 79
Bank Number 63
Burst count 68, 69
Burst Mode 83
Burst mode 68, 69
burst mode 12
Burst Mode Item Count 84

C

Calibration and Standardization Commands 40
calibration expert system 13
Calibration Mode 43
Calibration Mode of the Current Material Entry 42
Calibration Table in the Library 41
Chopper Speed 22
Clear the Current Burst Mode Item Count 85
Clear the current data series 66
Clear the Head Overheating Alarm 39
Clear the head temperature alarm I7CALM 4
COMPOSER 13
CONDITIONS OF GUARANTEE 2
Configuring 7
Cooler Enable Status 29
Cooler Linking Status 32
Cooler On/off Status 31
Cooler Status 33
Cooler Temperature 30
Cooling Enable 34
Cooling Linking 35
Copy the Temperature Series to Bank4 79
Current Batch Size 74, 75
Current Burst Size 80, 81
Current Library Name 88, 91
Current Material Entry 40

D

dark surface 12
Data Acquisition Commands 57
DATAEX 7

E

Expansion Module 90
Expansion module 13
expansion module 97
Expansion Module Number 96

F

Fast Fourier Transform 95
filter 15
Filter Characteristics 14, 15
frame type 10

G

General 8
General Commands 11
General System Status 11
Get Samples 76
Get the Burst Mode 83
Get the Current Material Entry Name 87
Get the Expansion Module Number 96, 98, 99, 100
Get the Expansion Module Signal 61
Get the Head Overtemp Status 38
Get the head temperature alarm status I7GALM 4
Get the Limit Switch Low Level 105
Get the Optical Head Temperature 57, 104
Get the Unit for Moisture 86, 93

I

I7AUTOOFF 42 69
I7AUTOON 41 68
I7BEEP 34 94
I7CALM 105 39
I7CBUC 117 0x75 85
I7CDPREP 106 0x6A 108
I7CLRSER 21 66
I7COPYT 98 79
I7DPACT 63 0x3F 100
I7DPDEACT 64 0x40 101
I7DPINIT 65 93
I7FFT 83 95, 108
I7G2STATUS 86 0x56 12
I7G3STATUS 89 13
I7GAINLOCK 51 19
I7GAINOPEN 52 20
I7GALM 104 38
I7GAMODE 59 71

I7GBANK 55 63
I7GBATCH 57 74
I7GBUC 116 0x74 84
I7GBUM 115 0x73 83
I7GBURST 113 0x71 80
I7GCOOLING 90 29
I7GCOOLINK 95 32
I7GCOOLON 94 31
I7GCOOLSTA 97 33
I7GCOOLTMP 93 30
I7GDPACT 66 0x42 99
I7GDPADR 61 0x3D 98
I7GETAUTO 43 65
I7GETDM 35 62
I7GETHI 32 0x20 104
I7GETLO 33 0x21 105
I7GETLOCK 53 16, 18
I7GETLPM 37 25
I7GETMAT 14 40
I7GETTIM 40 72
I7GETTMP 46 59
I7GETUSG 28 23
I7GFILTER 50 14
I7GFREQ 60 22
I7GLAMP 74 21
I7GLIBNM 29 88
I7GMATNM 31 87
I7GMODE 16 42
I7GNXMOD 109 0x6D 96, 101
I7GSHIFT 68 53
I7GSTATUS 76 11, 12
I7GSTD 73 54
I7GSTDM 71 52
I7GTLPF 101 0x65 16
I7GUNIT 13 86
I7GVOUT 88 27
I7GWEB 48 60
I7GWEB2 100 57, 58, 104, 105
I7GWEBB 103 36
I7GXMOD 108 61
I7GXNAME 110 0x6E 90
I7MOIST 11 0x0B 57
I7NOP 91 0x5B 102
I7RXMAT 27 44
I7SAMODE 58 70
I7SAMPLE 36 67
I7SBANK 54 64
I7SBATCH 56 75
I7SBUM 114 0x72 82
I7SBURST 112 0x70 81
I7SCOOLING 92 34
I7SCOOLINK 96 35

I7SDPADR 62 0x3E 103
I7SETHI 18 0x12 106
I7SETLO 19 0x13 107
I7SETLPM 38 26
I7SETMAT 15 41
I7SETTIM 39 73
I7SFILTER 49 15
I7SLIBNM 30 91
I7SMODE 17 43
I7SSHIFT 67 50
I7SSTD 72 51, 106, 107
I7SSTDM 70 56
I7STDZE 69 55
I7STERM 47 24
I7STLPF 99 17
I7SUNIT 12 92
I7SVOUT 87 28
I7SWEBB 102 37
I7SXCOM 111 97, 103
I7TEST 10 89
I7TXMAT 26 47
I7TXSER 20 76
Introduction and Taking into Use 7

K

Keyboard Mode 24

L

Lamp Status 21
linked autotimers 12
Locking 19
Locking Status 18
Low Power Mode 25
Low Power mode 26

M

Material Entry Number Used in Standardization 52
Memory Bank Commands 62
Meter's Identifier String 1 89
MULTI/QUICK 43

N

No Operation Command NOP 102
Number of Samples in the Current Bank 62

O

Offset for Standardization 50
Offset for Web Temperature 37
Offset Value Resulting from Standardization 53
Operating the Slave via Fieldbus 8

Optional Web Temperature 58
overheating of head 13
overheating of the head 38
overtemperature alarm 13

P

Passing Commands 8
Prepare Configuration Data Upload 108
procedure for sending 8

Q

quiet booting 12

R

Read the Calibration Table Entry 44
reflective surface 12

S

Schematic 109
Second System Status 12
Send a Command to the Expansion Module 97
session start 12
Set Profibus DP Slave Address 103
Set the Burst Mode 82
Set the Calibration Table Entry 47
Set the Profibus DP Inactive 101
Short Pulse to the LED Indicator 94
Standard Material Entry Number 56
Standard Moisture Value for Standardization 51, 106, 107
Standard Value Set for Standardization 54
Standardize 55

T

Take a sample 67
temperature autotimer 79
Terminal Mode 24
Text String Commands 80
Third System Status 13
Troubleshooting Hint 7

U

Unit for Moisture 92
Usage Counter 23

V

Voltage Output Source 27, 28

W

web break suspicion 13
web OK 12
Web Temperature Filter 16, 17
web temperature filter 13
Web Temperature Offset 36